Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Sistema basat en una placa Arduino i sensors de proximitat per al correcte moviment d'un vehicle motoritzat

MEMÒRIA

Autor: Joan Estrada Peñas
Director: Emili Lupon Rosés

Convocatòria: Abril 2018

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest projecte, basat en un microcontrolador integrat en una placa Arduino, consisteix en el desenvolupament d'un vehicle motoritzat que és capaç de circular per una trajectòria delimitada lateralment, de manera que sempre mantingui una distància igual a ambdós costats del vehicle. El sistema ha de detectar la distància que té a cada costat respecte a la paret, i modificar la trajectòria de manera que les distàncies s'igualin el més aviat possible.

Per a fer possible aquest projecte el primer pas ha estat la definició d'especificacions, com el nombre de sensors, el nombre de rodes, el nombre de rodes motoritzades i el sistema de control. Després s'ha fet el disseny de l'estructura del sistema i, a continuació, s'ha escollit el hardware del mateix. L'elecció dels components s'ha fet en base a la seva disponibilitat en el mercat, a la compatibilitat entre tots ells i al seu cost econòmic. Un cop establerts tots els components del hardware i la seva integració, s'ha procedit a la programació de la placa Arduino, que s'ha fet en el propi llenguatge de la marca basat en C++. El programa s'ha dissenyat per a què controli de manera ordenada, evitant interferències i conflictes elèctrics, tots els elements que estan connectats a la placa Arduino.

El sistema final permet al vehicle circular de manera autònoma per una trajectòria delimitada de manera que sempre se situï en el punt mig d'aquesta.

Aquest projecte ha assolit de manera satisfactòria els objectius proposats. La distància màxima de detecció de distàncies és d'uns 4 m.

Índex

1. Glossari

ALU (de l'anglès *Aritmetic Logic Unit*): és un circuit digital que realitza operacions aritmètiques i operacions lògiques amb un o dos operands.

Baud rate: és el nombre d'unitats de senyal transmeses per segon. Un *baud* pot contenir diversos bits.

CPU (de l'anglès *Central Processing Unit*): és el hardware dins un computador o altres dispositius programables, que interpreta les instruccions d'un programa informàtic mitjançant la realització de les operacions bàsiques aritmètiques, lògiques i d'entrada/sortida del sistema.

Driver: mòdul que permet controlar a través d'ell el funcionament d'altres mòduls.

EEPROM (de l'anglès *Electrically Erasable Programmable Read Only Memory*): és un tipus de memòria de només lectura que pot ser programada, esborrada i reprogramada elèctricament.

GND: de l'anglès Ground, serveix per designar els punts del sistema que estan connectats a la tensió de terra o massa.

GPIO (de l'anglès *General Purpose Input Output*): és un pin genèric d'entrada i sortida, en un circuit integrat,º que es pot controlar per l'usuari en temps real.

HDMI (de l'anglès *High Definition Multimedia Interface*): és una norma per a la interconnexió d'àudio i vídeo digital sense comprimir.

HDTV (de l'anglès *High Definition Television*): format que es caracteritza per emetre senyals digitals amb una qualitat superior als sistemes tradicionals analògics de televisió en color.

IDE (de l'anglès *Integrated Development Environment*): és una aplicació de software que proporciona serveis integrals per facilitar el desenvolupament de software al programador del microcontrolador.

Jack: connector d'entrada o sortida amb diverses mides universals.

Microcomputador: és un computador la CPU del qual és un microprocessador.

Microcontrolador: és un microcomputador materialitzat en un únic xip integrat (conté la CPU, la memòria de programa, la memòria de dades i els ports d'entrada/sortida), destinat a la implementació d'aplicacions relativament senzilles.

Microprocessador: és un processador o CPU materialitzat en un únic circuit integrat o xip. De vegades es requereixen dos circuits integrats per materialitzar microprocessadors molt potents, rebent el conjunt el nom de *chipset*.

NTSC (de l'anglès *National Television System Committee*): és un sistema de transmissió del senyal de televisió.

PAL (de l'anglès *Phase Alternating Line*): és un sistema de transmissió de senyals analògics de televisió en color amb l'objectiu de millorar la qualitat i reduir els errors de fase que es produeixen en el sistema NTSC.

PWM (de l'anglès *Pulse Width Modulation*): és una tècnica per obtenir resultats analògic utilitzant mitjans digitals. Es crea un valor analògic modulant l'amplada de polsos digitals, de tal manera que el valor mig resulti ser el valor analògic desitjat.

RAM (de l'anglès *Random Access Memory*): la memòria d'accés aleatori s'utilitza com a memòria de treball de computadores per al sistema operatiu, els programes i la major part del software. Materialitza la memòria de dades i, en ordinadors en els que els programes executats canvien al llarg del temps, la memòria del programa.

RISC (de l'anglès *Reduced Instruction Set Computer*): en arquitectura computacional, és un tipus de disseny de CPU utilitzat en microprocessadors que es caracteritza per tenir instruccions de mida fixa i presentades en un nombre reduït de formats, i perquè només les instruccions de càrrega i emmagatzematge accedeixen a la memòria de dades.

ROM (de l'anglès *Read Only Memory*): Memòria que només permet la lectura de dades un cop han estat gravades en ella.

SD (de l'anglès *Secure Digital*): és un dispositiu en forma de targeta de memòria per a dispositius portàtils.

SoC (de l'anglès *System on Chip*): és un tipus de dispositiu que integra en un sol xip els diferents components del sistema.

UART (de l'anglès *Universal Asynchronous Receiver-Transmitter*): és un mòdul digital que implementa la interfície d'algunes comunicacions en sèrie asíncrones.

USB (de l'anglès *Universal Serial Bus*): és un bus estàndard industrial que defineix el cablejat, els connectors i el protocol de comunicacions emprat en un bus de dades per a connectar, comunicar i alimentar perifèrics des dels ordinadors.

2. Prefaci

2.1. Origen del projecte

Des de fa un temps m'ha interessat cada vegada més el món de l'automoció i els importants canvis que està experimentant darrerament, sobretot en l'àmbit de l'electrònica. Cada vegada són més i més grans els reptes que es proposen les grans marques de vehicles ja sigui per fer la conducció més còmoda, més fluida o més segura.

En aquesta línia va sorgir la idea de fer un vehicle que fos capaç de mantenir sempre la mateixa distància amb un vehicle que anava davant seu. La idea podia ser aplicada en els cotxes per a funcionar, per exemple, en l'entrada de les grans ciutats, on són freqüents les arrancades i parades constants, que entorpeixen la fluïdesa del trànsit. No obstant, finalment, la idea va anar derivant cap al projecte actual a proposta del director del TFG, i em va semblar també de gran utilitat i interès de cara al futur. El projecte s'ha realitzat dins dels meus coneixements en electrònica i programació.

2.2 Motivació

La finalitat final del projecte és la realització del treball de fi de grau del GETI (Grau en Enginyeria en Tecnologies Industrials) de l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona. El treball està definit en la memòria dels títols de grau de l'Escola com un exercici original a realitzar de manera individual, que cal presentar i defensar davant d'un tribunal.

Amb aquest projecte també pretenia ampliar els coneixements de l'àrea de l'electrònica, a partir dels coneixements que vaig adquirir a l'assignatura d'Electrònica, assignatura troncal del GETI. Volia aprofundir més en l'estudi dels microcontroladors i dels diferents components electrònics que poden formar un sistema, així com en la seva interconnexió.

De retruc també he desenvolupat interès per l'aprenentatge i l'aprofundiment en la programació, ja que en el grau s'ensenya un llenguatge diferent que s'ha utilitzat al projecte i es fa només als primers cursos, fet que provoca que quedi una mica oblidat en arribar a les etapes finals del grau

3. Introducció

3.1 Objectius del projecte

El projecte consisteix en el disseny i realització física d'una maqueta d'un vehicle basada en un microcontrolador. El vehicle haurà de ser capaç de circular per una trajectòria delimitada mantenint sempre una distància igual a dreta i esquerra amb les parets que l'envolten, de tal manera que circuli per l'eix de la trajectòria.

L'adquisició de la distància es durà a terme mitjançant un sensors. A continuació es procedirà al processament de les mesures proporcionades per aquests, i a la seva transformació a un format que pugui ser entès pel sistema de control.

També es desenvoluparà un software per al microcontrolador, que permetrà dur a terme les funcions desitjades de cada component del sistema, així com la seva integració.

Un cop assolits els objectius principals, s'intentarà millorar el codi del microcontrolador per tal que la trajectòria sigui el més estable possible i sigui robusta davant l'entrada de canvis i pertorbacions.

3.2 Abast del projecte

En aquest apartat es descriuen les limitacions que s'han establert al projecte abans de començar i que han afectat al desenvolupament i al resultat del mateix.

En primer lloc, s'ha escollit Arduino com a microcontrolador i com a programa de llenguatge perquè l'estudiant ja havia realitzat un projecte en una altra assignatura de la universitat que funcionava amb aquest entorn, i era més senzill de reprendre que no pas haver de començar amb un llenguatge i un entorn completament nous. En conseqüència, s'han escollit components que fossin fàcilment compatibles amb Arduino i que fossin assequibles econòmicament per a un treball de fi de grau.

D'altra banda, es compta amb les limitacions pròpies del sistema de mesura dels sensors, les quals seran esmentades en apartats posteriors. Això farà que les distàncies a les que es puguin situar les parets seran petites i que no hi hagi canvis de direcció massa bruscs en les parets. A més, no es requerirà que el sistema tingui una autonomia en quant a funcionament molt elevada, ja que això encariria el cost del projecte i afegiria dificultats logístiques addicionals.

4. Especificacions del sistema

En aquest punt es definiran en detall quines són les característiques que ha de presentar el sistema final, com per exemple, el format en què s'han d'obtenir les distàncies, com ha de ser el codi del microcontrolador, amb quina freqüència s'han de mesurar les distàncies, a quina velocitat han de girar els motors, etc.

4.1. Nombre de sensors

És necessari definir de quants punts de la perifèria del vehicle es vol mesurar la distància a les parets exteriors. D'entrada amb 2 sensors n'hi ha prou, un que mesuri per cada costat. Si bé es cert que es podrien afegir un seguit de sensors més, que donessin una idea més exacta al vehicle d'allò que té al seu costat, aquest fet no és estrictament necessari pel que fa a aquest projecte en concret. Com que el cotxe només ha d'anar endavant i girar cap als dos costats, no és necessari incloure sensors a la part davantera del vehicle (per si calgués detectar un possible obstacle en la trajectòria) ni tampoc al darrera, per si calgués girar en sentit oposat. És cert que en una possible sofisticació del projecte, aquest sensors addicionals es podrien afegir.

4.2. Obtenció de la distància

La placa d'Arduino haurà d'obtenir la distància de cada sensor en mil·límetres i amb una precisió de 3 mm. Les distàncies podran oscil·lar entre 2 cm i 400 cm. A mesura que el microcontrolador vagi obtenint les distàncies, haurà d'anar comparant les dues mesures. Un cop comparades haurà de decidir cap a quin costat girar i amb quina velocitat.

4.3. Nombre de rodes

El nombre de rodes escollit per a realitzar el projecte ha estat 4. Es podria haver fet un sistema amb 2 rodes fixes al davant i una roda amb el moviment lliure al darrera, que només servís de suport. No obstant, el fet d'utilitzar 4 rodes li confereix estabilitat al vehicle i li permet tenir més opció de moviments. Les 4 rodes són fixes, és a dir, no és possible canviar la seva orientació. El gir s'aconsegueix modulant la velocitat de gir de les rodes.

4.4. Nombre de rodes motoritzades

D'entrada, amb 2 de les 4 rodes motoritzades, n'hi hauria prou per a fer funcionar el vehicle de la manera esperada. No obstant, el xassís definitiu en el que anaven muntades les rodes, venia acompanyat de 4 motors, un per roda. Aquests no permeten el gir lliure dels seus eixos. Per tant, totes les rodes estan motoritzades i el que s'ha fet es connectar els pols dels

dos motors del mateix costat al mateix senyal de control, de manera que les rodes d'un mateix costat sempre giren a igual velocitat, i això permet fer girar el vehicle.

4.5. Alimentació del sistema

L'objectiu del projecte és dissenyar un cotxe que funcioni de manera autònoma. Per tant interessa que el sistema es pugui alimentar de manera conjunta amb un sistema d'alimentació que no depengui de la xarxa elèctrica, tal com una bateria. La bateria ha de tenir el voltatge suficient per a què pugui alimentar tots els components sense superar les seves tensions d'alimentació màximes. Al mateix temps ha de tenir una capacitat suficientment gran per a què el vehicle pugui funcionar durant, com a mínim, 1 hora.

4.6. Cicle de funcionament

El microcontrolador haurà de llegir la distància de tots els sensors un cop cada cicle del programa. Un cop llegides, haurà de convertir les distàncies a mil·límetres i comparar-les, per veure de quina paret està més a prop. Una vegada detectat en quin punt es troba el vehicle, haurà d'aplicar un càlcul que transmeti el resultat als motors de les rodes, de tal manera que el vehicle s'allunyi de la paret la distància adequada que permeti situar-lo de nou al centre de la trajectòria.

5. Arquitectura del sistema

Amb el contingut presentat a la introducció i a les especificacions es pot dibuixar un esquema funcional del projecte. Els principals elements del sistema són: els 2 sensors, la placa Arduino, un driver controlador de motors, i les rodes de l'esquerra i de la dreta junt amb els seus motors, i tot això alimentat per una bateria. A la figura 1 es pot veure un esquemàtic del funcionament del sistema.

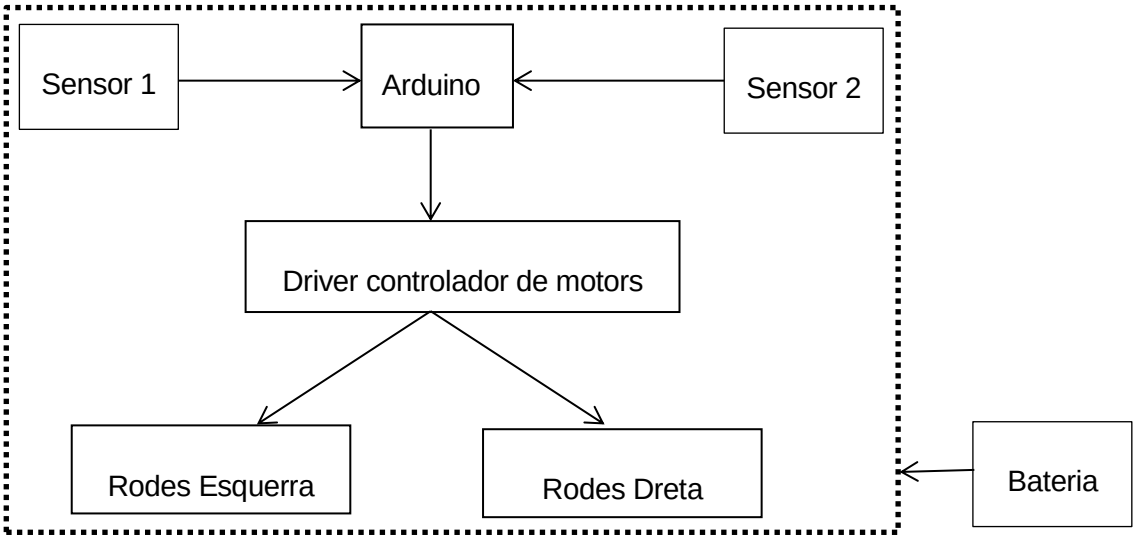


Figura 1: Esquema del sistema

6. El microcontrolador

Un microcontrolador és un circuit integrat programable, capaç d'executar les ordres gravades en la seva memòria de programa. Està format per diversos blocs funcionals, els quals compleixen una tasca específica. Un microcontrolador inclou en el seu interior les tres principals unitats funcionals d'una computadora: unitat central de processament, memòria i perifèrics d'entrada i sortida.

Quan es fabrica el microcontrolador, no conté instruccions a la memòria de programa. Després cal gravar-li a la memòria algun programa, el qual pot ser escrit en llenguatge assamblador o algun altre llenguatge per a microcontroladors. No obstant, perquè el programa pugui ser gravat finalment a la memòria, cal que sigui codificat en sistema numèric hexadecimal. La memòria *flash* emmagatzema de forma permanent, encara que no s'alimenti el microcontrolador, les instruccions que s'executaran.

A més a més de la memòria de programa, el microcontrolador generalment inclou també un generador de senyal de rellotge integrat, una petita quantitat de memòria de dades, de tipus memòria d'accés aleatori o *RAM* (*Random Access Memory*) i algunes vegades, també de tipus *EEPROM* (*Electrically Erasable Programmable Read Only Memory*). Usualment també disposen d'una gran varietat de dispositius d'entrada i sortida, tals com un convertidor analògic/digital, temporitzadors i *UART*s (*Universal Asynchronous Receiver-Transmitter*).

El generador de senyal de rellotge, és un component que produeix impulsos elèctrics binaris amb una determinada freqüència. És el que permet evolucionar al microcontrolador, que és un sistema seqüencial que requereix d'un senyal de rellotge periòdic. Ja que és necessari tenir-lo per realitzar aquesta tasca, després s'aprofita per a fer altres feines, com per exemple fer de coordinador de les accions de diversos circuits.

La memòria *EEPROM* és un tipus de memòria *ROM* (*Read Only Memory*). La memòria *ROM* és una memòria només de lectura que permet emmagatzemar dades. La memòria *EEPROM* pot ser programada, esborrada i reprogramada elèctricament.

La memòria *RAM* és la que fa servir el processador per a emmagatzemar els resultats dels càlculs matemàtics realitzats. S'anomena "d'accés aleatori" perquè es pot llegir o escriure en una posició de la memòria amb un temps d'espera igual per a cada posició, sense ser necessari seguir un ordre per accedir a la informació de la manera més ràpida possible.

La memòria *Flash* és una mena de memòria *EEPROM*, però amb un accés d'escriptura més ràpid. Usualment s'implementa amb una memòria *RAM* petita que fa d'interfície ràpida (accés ràpid) més una memòria *EEPROM* que emmagatzema les dades de la *RAM* (accés lent).

L'*UART* és un perifèric que facilita la transmissió en sèrie de dades de manera asíncrona. Està format per dos blocs independents: un emissor i un receptor, amb els pins TX i RX del microcontrolador, respectivament. Permet enviar i rebre dades de forma simultània, tenint els blocs emissor i receptor la mateixa velocitat. La velocitat està determinada per un circuit intern a l'*UART*, anomenat generador de *baud rate*. El *baud rate* és el nombre d'unitats de senyal transmeses per segon (en aquest cas cada unitat de senyal és un bit).

6.1 Alternatives

La gran oferta de microcontroladors en el mercat fa necessari l'anàlisi detingut de les diverses alternatives. En primer lloc, es distingiran dos grans grups: els microcontroladors que es venen per separat i els que venen ja integrats en plaques de desenvolupament.

Els que es venen per separat, moltes vegades necessiten anar incorporats en una placa de desenvolupament a posteriori, per facilitar-ne la connexió de sensors i actuadors i per a fer més àgil el procés de programació de la memòria.

En el cas del projecte, des d'un primer moment es volia utilitzar un microcontrolador que ja anés integrat en una placa de desenvolupament, i que a més disposés d'un entorn de programació senzill i intuïtiu. Al mateix temps, es buscava que tingués un cost accessible per al projecte. En aquest punt, la disjuntiva principal es trobava entre les plaques Arduino i les Raspberry Pi. Si bé Raspberry Pi no és un sistema basat estrictament en un microcontrolador, la seva orientació acadèmica i la facilitat de funcionament que presenta, la feien també una opció atractiva.

Encara que Arduino i Raspberry Pi siguin molt similars externament, en realitat són molt diferents. En primer lloc, Raspberry Pi és una computadora o microprocessador més potent implementat en varis xips, mentre que Arduino és bàsicament una computadora en la que tot està integrat en un xip, i això el converteix en un microcontrolador.

El microprocessador és un circuit integral complex d'un sistema informàtic. De manera anàloga al microcontrolador, és l'encarregat d'executar els programes, des del sistema operatiu fins les aplicacions d'usuari. La diferència principal rau en la quantitat de dades que pot processar i la velocitat en la qual ho pot fer.

Malgrat que l'Arduino pugui ser programat amb petites aplicacions com C, aquest no pot executar tot un sistema operatiu i certament no podrà ser el substitut d'una computadora per a les funcions en les que aquesta és requerida.

Els dos entorns van ser dissenyats originàriament per a funcionar com a eines per a l'ensenyament, motiu pel qual s'han tornat tan populars i són relativament senzills d'aprendre a utilitzar.

A continuació es farà una descripció més detallada de Raspberry Pi i d'Arduino, se'n compararan les característiques i s'explicarà quina ha estat la decisió final.

6.2 Comparativa de prestacions i selecció del microcontrolador

Raspberry Pi és una placa que treballa com una computadora basada en un microprocessador completament funcional. Utilitza llenguatges d'alt nivell com ara Python, C++ i Java. Utilitza un controlador *Broadcom*, que és un *SoC* (*System on Chip*) basat en un processador molt potent, l'*ARM11*, que té una freqüència de 700 MHz. Aquesta computadora no inclou un *display* (dispositiu que permet representar imatges), però pot ser connectada a un *display HDTV* i als estàndards de televisió *NTSC* o *PAL*. També conté un port Ethernet que permet connectar-la a la xarxa. Com a sistemes de desenvolupament d'aplicacions, es poden utilitzar ordinadors amb diferents sistemes operatius, com Mac, Windows i Linux, sempre i quan es tingui el software adient.

Figura 2: Placa Raspberry Pi

Arduino és una placa basada en un microcontrolador dissenyada per a aplicacions basades principalment en hardware. Està basada en un microcontrolador *ATmega328P*, que té una freqüència de 16 MHz.

Figura 3: Placa Arduino Uno

A continuació es mostra una taula amb una comparació de prestacions entre els dos entorns, a nivell quantitatiu. S'han escollit les plaques Arduino Uno Rev3 i la Raspberry Pi B.

Aquests dos models són els que més s'adapten als requeriments del projecte i entre les dues marques, són els models que més s'assemblen entre si.

	Arduino Uno R3	Raspberry Pi B
Processador	ATmega328P	Arm 11
Freqüència	16 MHz	700 MHz
Memòria *RAM*	2 KB	256 MB
Ports USB	1	2
Memòria de programa	32 KB	Inclosa a la *RAM*[1]
Memòria EEPROM	1 KB	Targeta SD
Tensió d'alimentació	7 – 12 V	5 V
Àudio	-	HDMI, Analògic
Vídeo	-	HDMI, Analògic
Ethernet	-	10/100
I/O	14 GPIO, 6-10 bits analógics	8 GPIO
Sistema operatiu	-	Linux
Entorn de programació	Arduino IDE	Linux, IDLE, Open-Embedded, QEMU, Scratchbox, Eclipse
Mesures	7,6 x 6,4 x1,9 cm	8,6 x 5,4 x 1,7 cm
Cost	26,58 €	40€

Taula 1: Comparativa entre Arduino i Raspberry Pi

[1] En la Raspberry Pi, tant la memòria de dades com la memòria de programa s'implementen amb *RAM* ja que hi ha un espai d'adreçament comú. En qualsevol cas, ha de tenir un *bootloader* en *ROM*.

Seguidament, comparant les especificacions tècniques i observant les principals funcions i aplicacions de les dues plataformes, s'ha pres la decisió d'utilitzar la placa Arduino. El principal avantatge d'Arduino és que disposa d'un microcontrolador i de múltiples entrades i sortides digitals, que el fa ser més apropiat per a la realització d'aquest projecte. Mentre que Raspberry Pi disposa d'un microprocessador més potent, aquesta potència no es requerida per a la complexitat d'aquest projecte. També disposa de molts més perifèrics encarats a aplicacions multimèdia, que en el cas d'aquest projecte tampoc són requerits. Ja que les prestacions d'Arduino són més que suficients, la diferència de preu ha acabat de declinar la balança.

En definitiva, quan es tracta de controlar aplicacions que involucren sensors, motors i actuadors, el sistema més apropiat és un microcontrolador. Si per contra, interessa fer un projecte que implica més complexitat en els càlculs, l'ús d'aplicacions multimèdia o el control de mecanismes sofisticats (com *displays*), és més convenient fer servir un microprocessador. Dit de manera col·loquial, un microcontrolador és un sistema més "muscular" (destinat a realitzar accions) i un microprocessador és un sistema més "cerebral" (encarat a càlculs), encara que ambdós sistemes puguin fer les dos coses, si més no són millors per a realitzar-ne una.

Un cop decidit l'entorn en el qual es realitzarà el projecte, cal fer un últim anàlisi dins de la marca Arduino i decidir quin dels seus models s'emprarà. Dins de les moltes plaques que Arduino oferta, se n'han considerat principalment quatre: Arduino Uno, Arduino Due, Arduino Mega i Arduino Nano. A la taula 2 es mostra una comparativa entre les principals característiques de cada placa.

Arduino	Uno	Due	Mega	Nano
Tensió de funcionament/ d'alimentació	5 V / 7-12 V	3,3 V / 7-12 V	5 V / 7-12 V	5 V / 7-9 V
Velocitat de la CPU	16 MHz	84 MHz	16 MHz	16 MHz
Pins Analógics In / Out	6 / 0	12 / 2	16 / 0	8 / 0
Pins Digitals In&Out / PWM	14 / 6	54 / 12	54 / 15	14 / 6
EEPROM (Kb)	1	-	4	1
SRAM (Kb)	2	96	8	2

Flash (Kb)	32	512	256	32
USB	Normal	2 Micro	Normal	Mini
UART	1	4	4	1
Preu[2]	20,00 €	34,00 €	35,00 €	20,00 €

Taula 2: Comparativa entre diferents plaques d'Arduino

Com es pot observar en el gràfic, Arduino Due i Mega són plaques molt més completes. No obstant, les característiques que aporten no eren crucials per al desenvolupament del projecte i el seu preu és considerablement més alt.

D'entrada, amb un rellotge del sistema de 16 MHz de freqüència és suficient per a dur a terme les tasques requerides a la placa. La placa Arduino Mega és similar a la Uno en aquest aspecte, però aporta més pins i més memòria (tant *EEPROM*, com *SRAM* com *Flash*). Per sort, el programa final no era tan extens com per a requerir aquestes especificacions de memòria, i el nombre de pins utilitzats no era tan gran com per sobrepassar els que proporciona Arduino Uno.

D'altra banda, la placa Arduino Nano té una forma molt més petita, i és només una placa de circuit imprès amb pocs perifèrics. Per a fer-la servir, cal col·locar-la en una protoboard per fer les connexions pertinents. S'ha descartat aquesta opció per tal de fer una distribució de l'espai més ordenada i senzilla. Com que les seves característiques són gairebé idèntiques a les d'Arduino Uno, també s'ha triat aquesta placa enfront de la Nano.

[2] Preus extrets de la web d'Arduino. L'adquisició final (apartat 13) de la placa ha estat diferent ja que venia d'un tercer proveïdor.

Figura 4: Arduino Uno

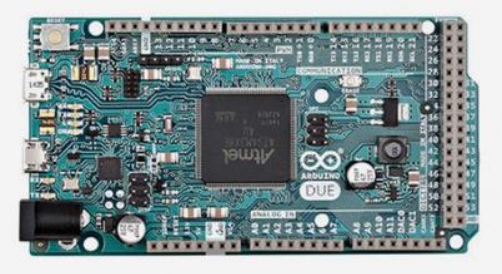


Figura 5: Arduino Due

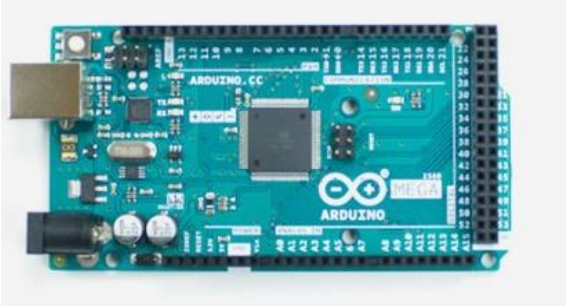


Figura 6: Arduino Mega

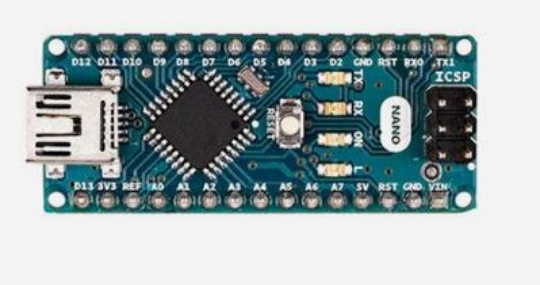


Figura 7: Arduino Nano

Així doncs, s'ha triat la placa Arduino Uno per a la realització del projecte per ser la més versàtil i la que millor s'adapta a les especificacions en termes de funcionalitat i costos.

6.3 Arduino Uno Rev3

L'Arduino és una plataforma electrònica de codi obert (*open-source*) composta de hardware i software per a dissenyar, desenvolupar i realitzar proves als prototipus d'electrònica i productes complexos. El hardware consisteix en un microcontrolador amb altres components electrònics que poden ser programats, utilitzant el software, per fer qualsevol tasca. La programació d'Arduino està basada en el llenguatge C/C++, que permet que el seu ús sigui fàcil per a qualsevol usuari aficionat a l'electrònica. Cal destacar que hi ha una gran varietat de llibreries desenvolupades per a moltes aplicacions.

6.3.1 Microcontrolador ATmega328P

Arduino Uno R3 està basada en un microcontrolador *ATmega328P* de la marca *Atmel*. Aquest microcontrolador d'alt rendiment és de tipus *RISC* (de l'anglès *Reduced Instruction Set Computer*). Els microcontroladors *RISC* tenen un joc d'instruccions reduït i senzill, són petits i tenen un consum, cost i temps de disseny reduïts. El microcontrolador d'Arduino realitza una instrucció cada 1 cicle del rellotge, amb una freqüència de rellotge de 16 MHz.

El model *ATmega328P* té una memòria de dades *SRAM* (de l'anglès *Static Random Access Memory*) de 2 Kbytes, una memòria de programa *Flash* de 32 bytes i una *EEPROM* de 1 Kbyte. D'altra banda, és un microcontrolador de 8 bits, i la seva *ALU* (*Arithmetic Logic Unit*), amb la que fa les operacions, és de 8 bits. La seva tensió d'alimentació pot estar compresa entre 1,8 i 5,5 V.

Mitjançant l'execució de poderoses instruccions en un sol cicle de rellotge, el dispositiu arriba a una resposta d'1 *MIPS* (Milions d'instruccions per segon), balancejant consum d'energia i velocitat de procés.

El microcontrolador incorpora un seguit de perifèrics, que permeten la seva interconnexió amb el món exterior. Els perifèrics més importants del microcontrolador i que s'utilitzen en aquest projecte són els canals *PWM* (*Pulse Width Modulation*), els temporitzadors (*timers*), els canals ADC i un port sèrie de recepció i transmissió de dades: l'*UART*.

A la figura 8 es mostra el diagrama de pins del microcontrolador.

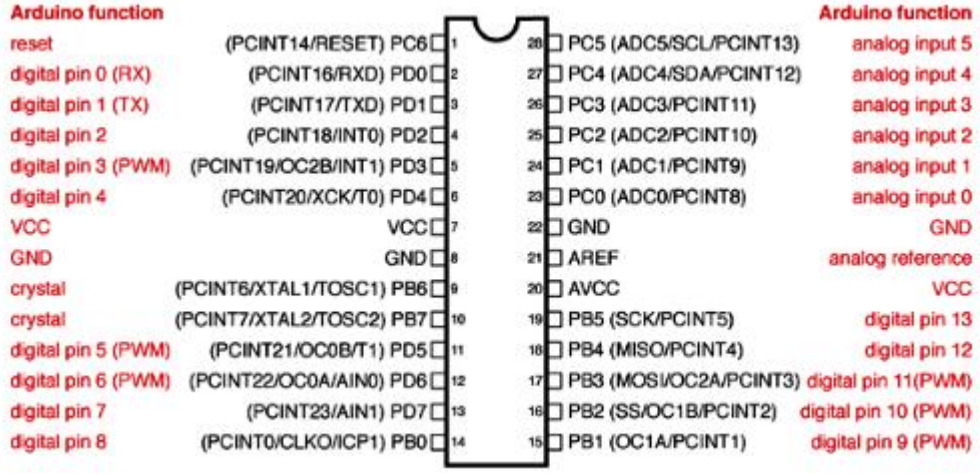


Figura 8: Diagrama de pins del microcontrolador ATmega328P

La placa Arduino està concebuda com a placa de desenvolupament d'aplicacions. Per això, juntament amb el microcontrolador, incorpora hardware que permet la connexió al microcontrolador d'altres elements de manera ràpida i senzilla, i facilita al seu torn el

desenvolupament d'aplicacions.

A més a més, disposa d'un entorn de desenvolupament integrat, *IDE* (*Integrated Development Enviroment*), que corre sobre un PC i que consisteix d'un editor per escriure els programes ("sketch") a executar en el microcontrolador de la placa hardware, d'un compilador i d'eines de comunicació amb la placa hardware.

Al compilar, el programa "sketch" es combina amb les funcions requerides de les llibreries estàndard de l'Arduino i les que han estat afegides per l'usuari. A continuació, el programa compilat és transferit al microcontrolador. Aquesta transferència es pot realitzar a través d'un cable USB o del port sèrie adequat del PC. L'Arduino conté un *ICSP* (*In-Circuit Serial Programming*) que permet als usuaris transferir el software al microcontrolador sense la necessitat d'extreure el circuit de la placa per posar-lo en un programador.

Entre els elements de connexió amb l'exterior més importants de la placa Arduino cal esmentar: pins digitals (alguns d'ells disposen de sortides de *PWM* que després s'esmentaran), pins analògics, pin AREF de referència analògica, botó reset (serveix per reiniciar la placa), pin VIN, pin 5V, pin GND, port d'entrada USB i *jack* d'entrada de 2,1 mm. A continuació s'explicaran les característiques més importants d'aquest perifèrics.

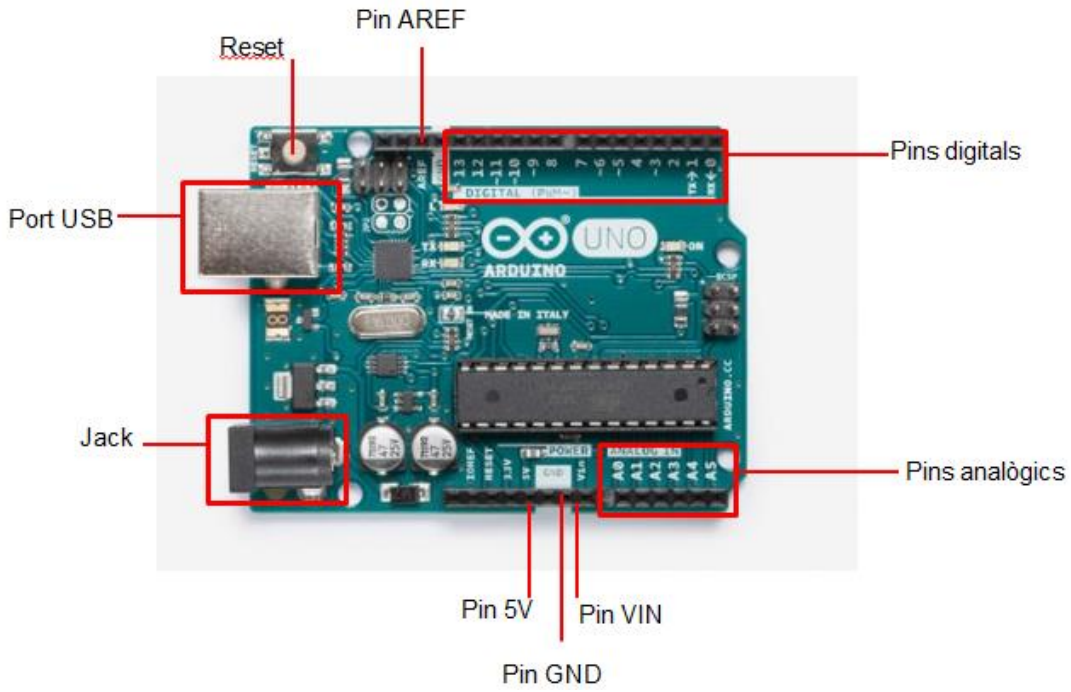


Figura 9: Perifèrics de la placa Arduino Uno R3

6.3.2 Pin VIN i jack

Aquest pin serveix com a punt d'alimentació externa per a la placa Arduino. Si es connecta el pol positiu d'una font d'alimentació externa (tal com una bateria) al pin VIN i el pol negatiu al pin GND, la pròpia placa regularà la tensió a 5 V. Això ho fa gràcies al regulador de tensió *NCP1117-D* que duu incorporat, la caiguda de tensió mínima del qual és de 1,2 V. Aplicant una tensió d'entre 6,2 i 12 V, el regulador retorna un potencial d'entrada a la placa de 5 V. Es poden aplicar fins a 20 V de tensió, però el fabricant recomana no sobrepassar els 12 V. En la mesura que major sigui la tensió, major serà el calor que el regulador haurà de dissipar, fent menys eficient el funcionament del sistema. Proporcionar més de 20 V farà malbé el regulador i la placa immediatament.

El jack és un port d'alimentació DC circular de 2,1 mm de diàmetre amb el pin positiu al centre. Si s'alimenta la placa a través del *jack* entre 7 i 12 V, el pin VIN té la capacitat de proporcionar una tensió de sortida similar a la del *jack* i un corrent una mica inferior (degut al consum de la placa). Això és gràcies a que internament, el pin VIN i el *jack* estan connectats.

A continuació, es pot veure una part de l'esquemàtic de la placa Arduino. El mòdul x1 anomenat POWERSUPPLY_DC21MMX correspon al jack d'entrada. D'aquest en surt una pista anomenada PWRIN, que conté la tensió d'entrada que se li aplica al jack. Després de passar pel díode D1, la tensió passa a ser VIN. En aquest díode, la caiguda de tensió serà de l'ordre de 0,7 – 0,8 V. Per tant, la tensió que sortirà pel pin VIN serà igual a la tensió que s'apliqui pel jack menys la caiguda de tensió en el díode.

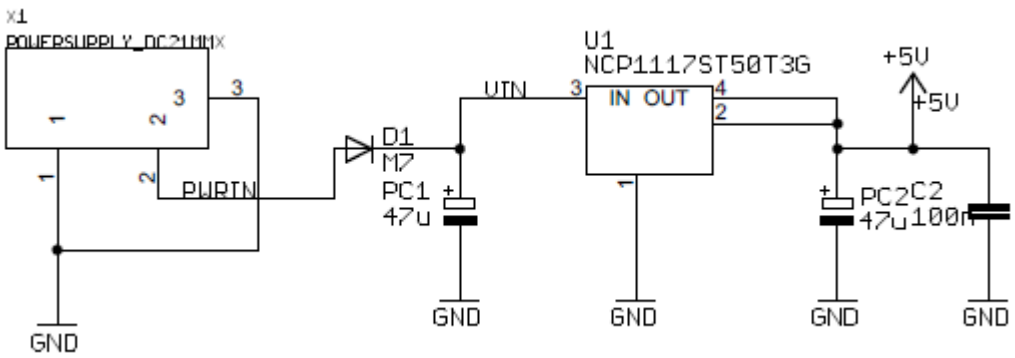


Figura 10: Esquemàtic de la part d'alimentació de la placa Arduino que inclou el jack

6.3.3 Pin 5 V

Aquest pin té bàsicament dos funcions. Es pot utilitzar com a pin d'alimentació de la placa si s'alimenta exactament a aquesta tensió. Alimentar directament a través d'aquest pin implica

saltar-se (*by-pass*) el regulador de tensió, i qualsevol salt inesperat en la tensió podria fer malbé la placa. Per altra banda, si s'alimenta la placa a través del *jack* entre 7 i 12 V, mitjançant el port USB a 5 V, o s'alimenta a través del pin VIN entre 6,2 i 12 V, aquest pin pot alimentar un altre component amb una tensió regulada de 5 V. En el nostre cas, s'ha usat aquest pin per alimentar els sensors *HC-SR04*.

6.3.4 Pins d'entrada analògics i pin AREF

La placa disposa de 6 pins d'entrades analògiques (A0-A5) les senyals de les quals es poden convertir a digitals amb 10 bits de resolució. La tensió que poden mesurar va des dels 0 fins als 5 V, encara que es possible canviar aquest rang utilitzant el pin AREF.
Per comprendre millor la utilitat del pin AREF, cal entendre com funciona un convertidor A/D. En el cas d'Arduino, com que disposa d'un convertidor de 10 bits, significa que la reva resolució és de 2^{10} = 1024 possibles valors. La tensió de referència de la mesura és de 5 V, que és la tensió de referència d'Arduino. Això implica que cada esglaó de mesura del convertidor és de 5 V / 1024 = 0,00488 V, una mica menys de 5 mV, fet que comporta que no pot discernir valors de tensió la diferència entre els quals sigui menor que aquest valor.

En definitiva el que fa el convertidor és proporcionar una comparació quantificada en relació a un valor de referència. En una situació ideal, la tensió de comparació contra la que es fa la comparació hauria de ser el valor màxim que pot presentar la possible senyal d'entrada, ja que així és quan s'aconsegueix la major resolució possible. Així doncs, el pin AREF ens permet establir una tensió de referència qualsevol per optimitzar més la discretització en amplitud de les entrades analògiques. D'altra banda, el convertidor d'Arduino té un temps de mostreig de 112 µs, que equival a una freqüència de mostreig de 8,928 KHz. Això vol dir que cada 112 µs, el convertidor llegeix el senyal analògic en qüestió i en fa la conversió a digital. Els canvis de tensió que només estiguin separats en el temps menys de 112 µs, no seran detectats ni convertits.

6.3.5 Pins digitals

Els pins digitals (1-13) poden servir tant d'entrada com de sortida a la placa, però no de manera simultània. En el disseny del programa s'ha d'especificar en quina direcció es vol que funcionin aquests pins. Aquest pins només poden emetre dos estats de tensió: "LOW" i "HIGH". Els dispositius connectats a ells interpreten els senyals digitals i els processen de manera adient.

Alguns dels pins digitals també poden emular una sortida analògica mitjançant la regulació de l'amplada dels polsos, també anomenat *PWM* (*Pulse Width Modulation*). Aquest fet

consisteix en activar la sortida digital durant un període de temps (*HIGH*) i desactivar-la durant la resta (*LOW*). La mitjana de la tensió de sortida, al llarg del temps, serà la tensió analògica desitjada. Aquesta tensió "analògica" és la que s'utilitza per fer moure els motors. Realment els motors estan alimentats durant un temps i no alimentats durant un altre, però la commutació és tan ràpida comparada amb la inèrcia dels motors, que a simple vista aquests no deixen de girar. Cada pin amb capacitat *PWM* té associat un perifèric, que és qui gestiona les temporitzacions (temps que està en *HIGH* i temps que està en *LOW*). Aquest perifèric només pot admetre valors en codi Arduino entre 0 i 255. El valor 0 donarà lloc a una sortida constant *LOW*, que equival a una sortida constant a 0 V. El valor 255 donarà lloc a una sortida constant *HIGH*, que equival a una sortida constant a 5 V.

6.3.6 Port USB

Ja s'ha vist anteriorment que l'*UART* és el port sèrie hardware que tots els microcontroladors tenen i que la comunicació sèrie és la base de la majoria de les comunicacions dels microcontroladors.

L'USB (Universal Serial Bus) serveix per connectar Arduino a l'ordinador, i la connexió de l'USB amb el microcontrolador es fa a través del port sèrie de l'*UART*. A la placa Arduino es fa servir el port sèrie USB per a tres funcions: fer servir el port com a alimentació regulada de 5 V a la placa, carregar el programa ja compilat a la memòria flash i connectar-se al port sèrie *UART* per comunicar-se amb el microcontrolador durant l'execució del programa. Això permet representar valors i dibuixar gràfics per entendre bé el funcionament d'algun dels components del programa.

7. Els sensors de distància

A continuació s'explicaran els diferents sensors que permeten calcular distàncies, i se'n compararan les prestacions per finalment escollir un model que permeti assolir els objectius del projecte.

7.1 Alternatives

A grans trets, existeixen en el mercat dos tipus de sensors que permeten calcular distàncies. Malgrat que la finalitat és la mateixa, aquests dos tipus de sensors disten molt en la manera base de funcionar. Els dos utilitzen un sistema de mesura similar, que consisteix en enviar un senyal i esperar a que torni, i un cop ha tornat, realitzar els càlculs adients per trobar la distància. La diferència principal rau en el tipus de senyal físic que envien: uns emeten senyals lluminosos i els altres senyals acústics.

Els primers, s'anomenen sensors d'infraroigs, ja que emeten ones electromagnètiques d'una longitud d'ona tal que pertanyen a l'espectre dels infraroigs. Els segons s'anomenen sensors d'ultrasons o ultrasònics, i emeten ones electromecàniques per sobre de la freqüència que l'oïda humana pot captar, d'aquí el seu nom.

Ja que la llum es propaga per l'espai a una velocitat molt més elevada que el so, els sensors d'infraroigs, en general, són més ràpids a l'hora de calcular distàncies. Cal entendre bé què s'està dient quan es menciona la paraula "calcular". El càlcul de la distància el fa el microcontrolador, en aquest cas l'Arduino. Això no depèn del tipus de sensor. El sensor l'únic que fa és enviar un senyal i esperar a que aquest torni, i un cop ha tornat efectuar els càlculs corresponents. La velocitat de càlcul serà un factor que dependrà del microcontrolador, no obstant, si s'agafa com a temps de càlcul el moment comprès entre que el sensor envia el pols de sortida i el moment en el que el microcontrolador retorna el valor de la distància, aquest temps serà menor en els sensors d'infraroigs, ja que la primera part que involucra al sensor serà més ràpida. No obstant, de vegades el funcionament intern dels sensors fa que el sensor d'infraroig no sigui més ràpid que el d'ultrasons, ja que les constants internes de temps són més grans.

Dins dels dos tipus de sensors, se n'han trobat dos (un de cada tipus) que són especialment atractius per al projecte. Són sensors relativament barats, disponibles al mercat, i senzills d'utilitzar. El sensor d'infraroigs que s'ha considerat és el *GP2Y0A21YK* de la marca *Sharp*. El sensor ultrasònic que també ha esta objecte d'anàlisi ha estat l'*HC-SR04*, molt disponible al mercat i de hardware obert (*open-source*).

Figura 11: Sensor HC-SR04 (esquerra) i sensor GP2Y0A21YK (dreta)

7.2 Comparativa de prestacions i selecció dels sensors

Per a decidir quin dels dos sensors compleix de manera més satisfactòria els objectius inicials del projecte, es mostra a continuació una taula amb una comparativa de les prestacions principals d'aquests dos mòduls.

	HC-SR04	*GP2Y0A21YK*
Tensió d'alimentació	5 V	4,5 – 5,5 V
Corrent de consum	15 mA	30 - 40 mA
Distància mínima	2 cm	10 cm
Distància màxima	400 cm	80 cm
Precisió en les mesures	0,3 cm	-
Preu	0,63 €	11,74 €

Taula 3: Comparativa de prestacions dels sensors

Pel que fa al tema de consums i tensions, tal i com es veurà en l'apartat 9.1, els dos compleixen amb les condicions que permetrien alimentar el sistema de manera autònoma. Si bé el sensor ultrasònic consumeix la meitat de corrent que el d'infraroigs, aquest fet no resulta decisiu ja que com es veurà en l'apartat 9.2, la capacitat de la bateria cobreix amb escreix el temps d'autonomia desitjat (veure apartat 4.5).

Un fet que si és important a l'hora de decidir, són les distàncies mínima i màxima de detecció. En aquest sentit, el sensor ultrasònic supera en els dos camp a l'altre. Interessa que el sensor pugui detectar una distància molt petita a la paret. Si de sobte hi ha un canvi

brusc en la trajectòria o una instrucció del software que obligui al vehicle fer un gir sobtat, el sensor ha d'estar preparat per a reaccionar si el vehicle s'ha apropat massa a la paret. Si s'utilitzés el sensor d'infraroigs, és possible que arribats a una distància menor que 10 cm de la paret, el vehicle es desorientés i col·lisionés contra ella. D'altra banda, la distància màxima de 80 cm pot ser suficient per al tipus de recorregut que s'intentarà recórrer en el projecte, però mai està de més tenir capacitat per a mesurar més distància. A més, el sensor d'infraroigs comença a perdre precisió en les mesures a mesura que la distància s'apropa als 55 cm, ja que les tensions de sortida són molt semblants entre si, tal i com mostra la figura 12.

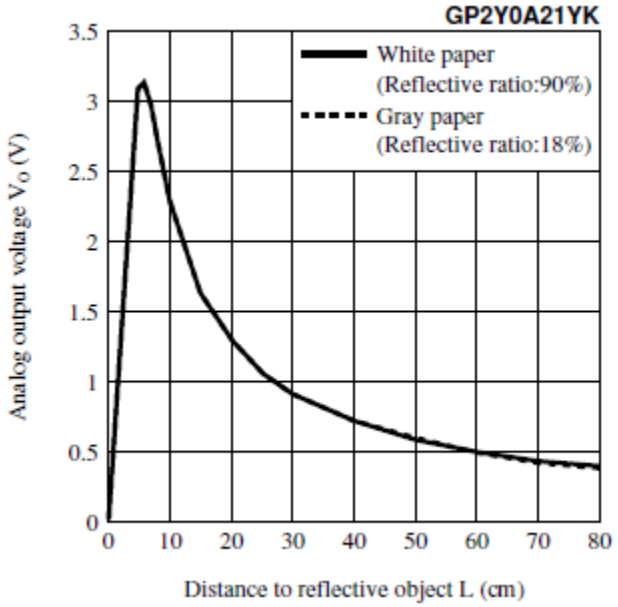


Figura 12: Tensió de sortida del sensor GP2Y0A21YK en funció de la distància al objecte

Pel que fa a la precisió de les mesures, el sensor ultrasònic *HC-SR04* és capaç de discernir distàncies que difereixin entre si 0,3 cm com a mínim. En canvi, per al sensor d'infraroigs *GP2Y0A21YK*, el fabricant no especifica a la fulla de característiques quina és la seva precisió. Si s'adquirís aquest sensor, caldria fer-ne primerament una prova de calibratge, ja que no es pot tenir un sensor precís sense calibrar-lo. Per tant, en aquest sentit, el sensor d'ultrasons afegeix simplicitat al sistema a l'hora de muntar-lo, ja que no té necessitat de calibratge, i el d'infraroigs sí. A més, sensor d'infraroigs pot presentar diferències en les mesures influenciat per la reflectivitat de l'objecte detectat.

Finalment, un factor molt decisiu ha estat el preu. Com es pot observar en la taula, el preu dels sensors *HC-SR04* és molt inferior al preu dels sensors *GP2Y0A21YK*, i com que també es busca que el projecte assoleixi els objectius al menor cost possible, s'ha acabat triant el sensor ultrasònic com a mòdul per a obtenir la distància a les parets del voltant.

7.3 Sensors d'ultrasons *HC-SR04*

7.3.1 Característiques generals

El sensor ultrasònic *HC-SR04* proporciona una mesura de la distància en un rang comprès entre 2 cm i 400 cm, i el nivell de precisió d'aquestes mesures s'acosta fins als 3 mm. Com ja s'ha esmentat anteriorment, el sensor consta d'un emissor ultrasònic, un receptor i un circuit de control. El principi bàsic de funcionament és el següent:

1. Establir l'interruptor del pin "Input Trigger" a nivell alt de senyal.
2. El mòdul automàticament envia un seguit de polsos i posa el pin "Output Echo" a nivell alt.
3. Si el senyal torna, el temps que triga en arribar és el senyal que es transmet pel pin "Output Echo".

El sensor consta de 4 pins de connexió: Alimentació de 5 V DC, "Input Trigger", "Output Echo"frfr11 i 0 V de terra o massa. Tot i així, les tensions d'alimentació poden variar des de 4,5 V fins a 5,5 V. El corrent que consumeix és de 15 mA.

7.3.2 Rang de mesura, exactitud i precisió

Per a qualsevol sensor direccional que funciona sense contacte físic la geometria del món exterior es determina a través del seu Camp de Visió(CDV) i del rang.

El CDV és una mesura del rang angular dins del qual el sensor és capaç de percebre objectes. Fora d'aquest rang els objectes no són detectables. El CDV es calcula normalment com la grandària angular del con o piràmide de visió, expressat en graus horitzontals i verticals.

El rang és la distància, prenent com a punt de referència el sensor, des del pla més proper en el que els objectes són detectats fins al més llunyà.

La resolució espacial, o Camp Instantani de Visió (CIDV), és el detall més petit que pot ser mesurat dins del CDV.

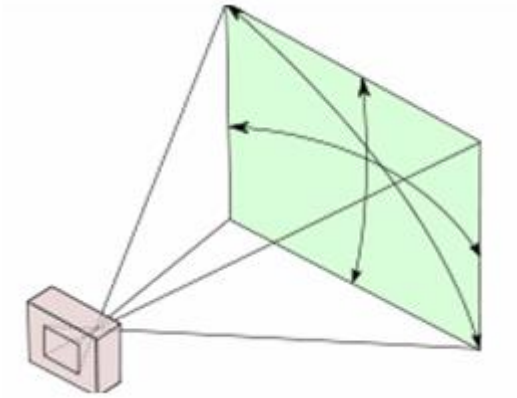


Figura 13: Representació del con de visió format pels camps de visió

Pel que fa a l'exactitud: "l'exactitud d'un sistema de mesura és quant a prop es troba el valor mesurat del valor real". La fidelitat (*precision* en anglès) es refereix a "la dispersió del conjunt de valors obtinguts de mesures repetides d'una magnitud".

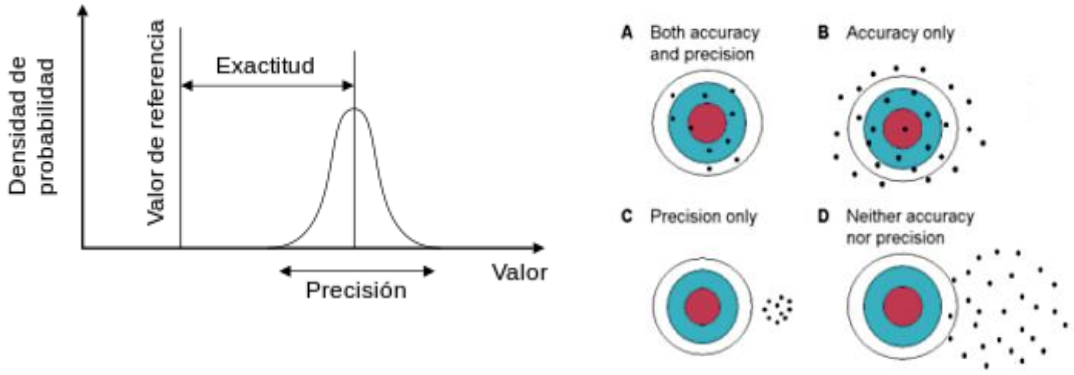


Figura 14: Explicació visual de l'exactitud (accuracy, en anglès) i de la fidelitat (precision)

En diversos estudis realitzats s'ha determinat el CDV, la resolució, la fidelitat i l'exactitud del sensor *HC-SR04*, i s'ha trobat que són una mica diferents a les especificades pels fabricants.

El camp de visió horitzontal (CDVH) varia entre els 18º i els 24º, amb un valor mig de 21º. El CDVH sembla estar una mica desviat cap a un costat, probablement degut a la asimetria del mòdul: el CDVH total és 9,5º+11,5º. Pel que fa al camp de visió vertical (CDVV) és aproximadament de 4º.

En quant a la resolució del con de visió, és aproximadament d'entre 0,6º i 1,4º. L'error absolut en l'exactitud pel que fa a les mesures és d'uns 0,035 cm/cm. Per últim, la fidelitat té una desviació estàndard d'entre 0,1 i 0,5 cm.

7.3.3 Diagrama de temps

A sota es mostra el diagrama de temps del sensor. Només es necessita subministrar un pols curt de 10 µs al pin "Input Trigger" per començar la mesura, llavors el mòdul emetrà 8 ràfegues cícliques ultrasòniques de 40 kHz i posarà a nivell alt el pin "Output Echo". Quan el sensor rebi de nou el senyal, el pin "Output Echo" passarà a nivell baix, subministrant a la placa Arduino el temps que ha estat en estat alt, que és el mateix que el temps que ha trigat l'ona sonora en anar i tornar fins a l'objecte detectat. A continuació, dividint el temps entre 2 (ja que només compta l'anada) i fent un càlcul de conversió a través de la velocitat del so, es troba la distància.

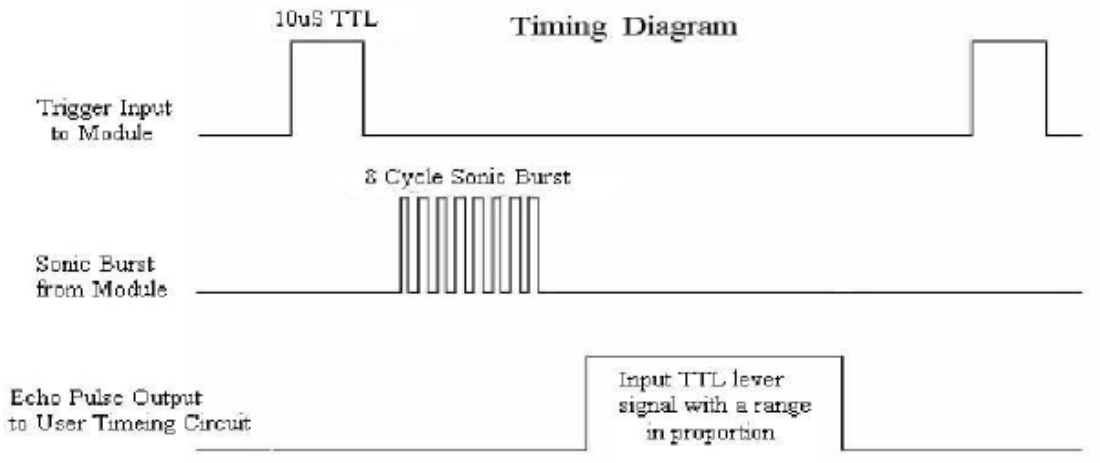


Figura 15: Diagrama de temps del sensor ultrasònic HC-SR04

8. Els motors i els seus drivers

A continuació s'esmentaran les diverses opcions que es van considerar a l'hora de triar els motors que mourien el vehicle i dels drivers que permetrien controlar-los.

8.1 Alternatives

8.1.1 Alternatives per al driver controlador de motors

Un driver controlador de motors és un mòdul electrònic, generalment integrat en una mateixa placa base, que permet controlar el sentit i la velocitat de gir d'un o més motors.

No es va trigar gaire en triar la opció definitiva pel que fa al driver utilitzat al projecte. Hi havia principalment dos opcions: el driver *L298N* de *STMicroelectronics* i el *Motor Shield for Arduino v2* del fabricant *Adafruit*.

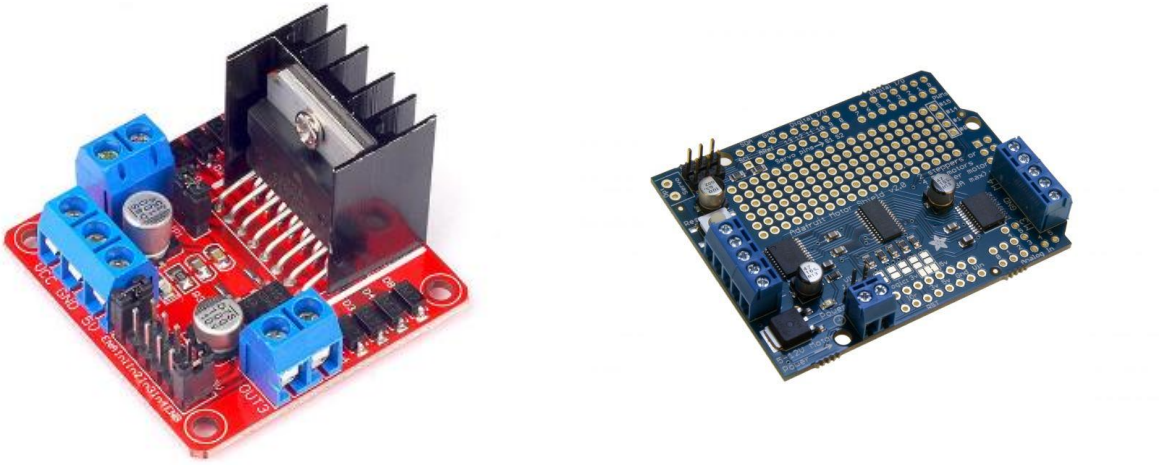


Figura 16: Driver L298N (esquerra) i Motor Shield v2 d'Adafruit (dreta)

El mòdul d'*Adafruit* és un sistema molt més complet que el driver *L298N* i que permet la connexió de fins a 4 motors independents. El problema que presenta és que ocupa molts pins de la placa Arduino, i aquests queden inservibles per a altres mòduls, com els sensors ultrasònics. Aquest problema es podria solucionar amb una placa Arduino Mega, que conté més pins digitals que la Arduino Uno, però això encariria el preu del projecte, com s'ha vist en l'apartat 6.2.

A més, també s'encariria el projecte a causa dels controladors de motors. El preu del *Shield* d'*Adafruit* és de 19,95 €. En canvi, el preu del driver *L298N* és de 1,58 €. Si la necessitat de tenir 4 motors independents fos fonamental, s'hauria optat per la via del mòdul d'Adafruit, però com que en el projecte, només és estrictament necessari que els motors de la dreta

siguin independents dels motors de l'esquerra, s'ha descartat aquesta opció per l'encariment que suposa, i per l'encariment d'haver de comprar una placa d'Arduino amb més pins digitals.

Així doncs, ja que la placa d'*Adafruit* era molt més cara que el driver *L298N* i no aportava millores considerables en termes de consum, eficiència i funcionalitat, finalment s'ha decidit optar per la utilització del driver controlador de motors *L298N*. A l'apartat 8.3 se'n fa una explicació més detallada.

8.1.2 Alternatives per als motors

Per a poder moure el projecte cal, com és lògic, la utilització de motors que permetin girar a les rodes. Com que el projecte és una maqueta d'un vehicle, s'han buscat diferents tipus de motors petits que permetin ajustar la velocitat desitjada per l'usuari. Els principals motors que es poden utilitzar en un prototipus demostrador podrien ser els següents: motors DC, motors pas a pas i servomotors de posició.

Tenint en compte que s'ha escollit treballar amb la placa Arduino, es fa una cerca de les especificacions tècniques dels diferents tipus de motors compatibles amb aquesta placa.

Primerament, els motors de corrent contínua (DC, de l'anglès *Direct Current*), com altres motors elèctrics, converteixen l'energia elèctrica en energia mecànica. Estan formats per dues parts, l'estator i el rotor. Considerant un rotor amb debanats i un estator amb debanats o imants permanents, quan el corrent elèctric circula pel debanat del rotor, es crea un camp electromagnètic. Aquest interactua amb el camp magnètic creat per l'estator, provocant que els dos pols d'igual signe dels camps magnètics es rebutgin, creant un parell de força que fa girar el rotor en un sentit concret. Si es vol canviar el sentit de gir del rotor, s'ha de canviar el sentit del corrent que circula pel rotor. Per a fer-ho, només cal invertir la polaritat de la pila o bateria amb la que s'alimenta el debanat del rotor.

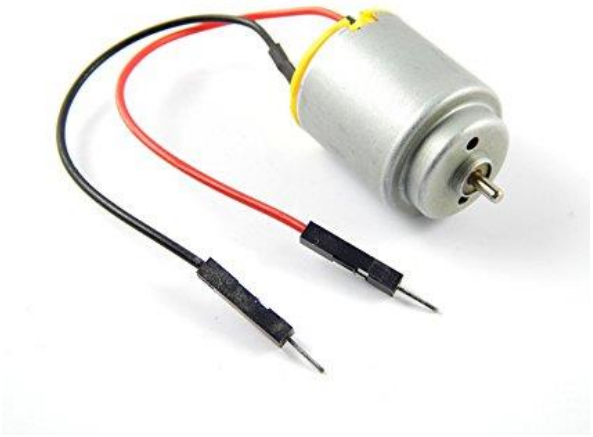


Figura 17: Motor DC

En segon lloc, els motors pas a pas (*stepper motor*, en anglès) són dispositius electromagnètics que converteixen impulsos elèctrics en moviments mecànics de rotació. La principal característica d'aquests motors és que es mouen un cert angle, anomenat pas, per cada impuls de corrent que reben. Normalment els passos poden ser de 1,8º a 90º per pas, depenent del motor. Són motors molt precisos, que poden quedar fixos en una posició, com els servomotors, i també són capaços de girar lliurement en un sentit o en un altre, com els motors DC. Els motors pas a pas també estan formats per un estator i un rotor, que van muntats sobre un eix.

Figura 18: Motor pas a pas

Hi ha tres tipus de motors pas a pas: imant permanent, reluctància variable i híbrid.

Els motors pas a pas d'imant permanent permeten mantenir un parell diferent de zero quan el motor no està alimentat. Depenent de la construcció del motor, és típicament possible obtenir passos angulars de 7.5º, 11.25º, 15º, 18º, 45º o 90º. L'angle de rotació és determina pel nombre de pols en l'estator.

Els motors pas a pas de reluctància variable, tenen un rotor multipolar de ferro i un estator debanat, opcionalment laminat. Rota quan la dent (o dents) més propera del rotor és atreta a la bobina de l'estator alimentada (obtenint-se, per tant, la ruta de menor reluctància). La resposta d'aquest tipus de motor es molt ràpida, però la inèrcia permesa en la càrrega és petita. Quan els debanats no estan alimentats, el parell estàtic d'aquest motors és nul.

Els motors pas a pas híbrids es caracteritzen per tenir diverses dents a l'estator i al rotor, i el rotor amb un imant concèntric magnetitzat axialment al voltant del seu eix. Es pot veure com aquesta configuració és una barreja dels motors de reluctància variable i els d'imant permanent. Tenen una alta precisió i un alt parell, i es poden configurar per a subministrar un pas angular tant petit com 1,8º.

En tercer lloc, els servomotors (coneguts col·loquialment com “servo”), són un sistema complex basat en un motor de corrent contínua que permet assolir una posició indicada dins del seu rang d’operació i mantenir-se fix en ella. Normalment l’angle pot variar de 0º a 180º i s’alimenta amb una tensió mínima de 5 V.

El servomotor està format per un motor de corrent contínua, una caixa reductora, un joc d’engranatges, un potenciòmetre i un circuit de control, on té integrat un “pont en H”. Pot aguantar una quantitat determinada de pes a través del parell del servomotor. Per a controlar-lo, s’utilitzen tècniques de PWM (modulació per amplada de polsos). La majoria d’ells treballen a una freqüència de 50 Hz (període de 20 ms). Quan s’envia un pols, l’amplada d’aquest determina la posició angular del servomotor. L’amplada varia segons el servomotor, però acostuma a estar compresa entre 0,5 i 2,5 ms.

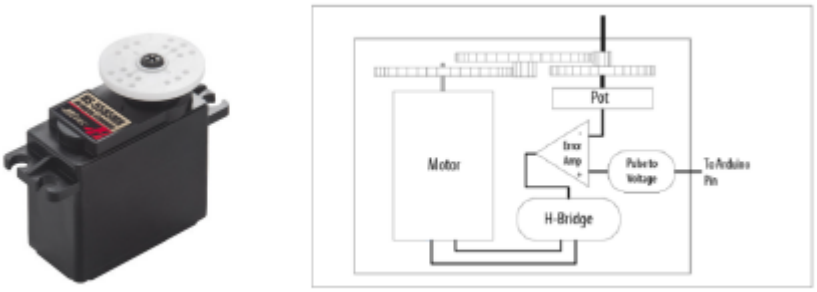


Figura 19: Servomotor i esquema de blocs del servo

8.2 Comparativa de prestacions i selecció dels motors

Per tal de decidir quin dels tres tipus de motors esmentats a l’apartat 8.1.2 s’utlitzarà, es mostra a continuació una taula on es poden veure les principals característiques de cadascun d’ells. Només es mostren característiques de motors que a priori puguin encaixar amb l’estructura del projecte. No tindria gaire sentit analitzar tota l’extensa gamma de motors dels tres tipus que hi ha al mercat

	Motor DC	Motor pas a pas	Servomotor
Tensió d’alimentació	3 – 6 V	8 - 12 V	4,8 – 6 V
Corrent consumit	100 mA	350 mA	250 mA[3]

[3] En funció de la velocitat de gir i de la càrrega de l’eix.

Velocitat de gir	240 RPM	200 RPM[4]	62,5 RPM
Preu	1 €	17 €	16 €

Taula 4: Comparativa dels diferents tipus de motors

Finalment, s'ha decidit utilitzar motors DC per a la realització del projecte. Les seves característiques de consum i tensió, els fan perfectament compatibles amb Arduino. A més també hi ha molts motors amb eixos ja pensats per a ser encaixats en rodes petites, i que també incorporen un sistema de muntatge senzill a un xassís que té les mides apropiades per a projectes amb Arduino. La seva velocitat és l'adequada per a la maqueta del vehicle. I finalment, el seu baix preu els converteix en candidat ideal.

8.3 Driver controlador de motors *L298N*

El driver *L298N* de *STMicroelectronics* es un mòdul que es pot emprar per controlar la direcció i la velocitat de gir dels motors del vehicle. Es pot usar per a motors que tenen una tensió nominal d'entre 5 i 35 V, i pot proporcionar un corrent de fins a 2 A per cada motor. Està basat en el sistema anomenat "pont en H".

El "pont en H" és un circuit electrònic que s'usa generalment per permetre a un motor de corrent continu girar en ambdós sentits, és a dir, endavant i endarrere. El terme prové de la típica representació gràfica del circuit. Es construeix mitjançant 4 interruptors, ja siguin mecànics o electrònics (transistors). Quan els interruptors S1 i S4 (veure figura 20) estan tancats, s'aplica una tensió positiva al motor, fent-lo girar en un sentit. Obrint aquests dos interruptors i tancant l'S2 i S3, la tensió s'inverteix i el motor gira en sentit contrari a l'anterior.

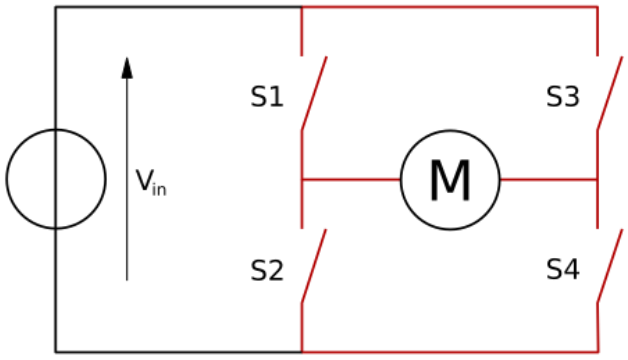


Figura 20: Esquema de funcionament del "pont en H"

[4] Variable en funció del nombre de passos, l'angle girat per cada pas, i la velocitat per fer un pas.

A la figura 21 es pot veure com el xip del driver està format per dos “ponts en H”. Cada pont presenta dues sortides, anomenades OUT que serveixen per subministrar la tensió als motors. Al haver-n’hi 4, es poden connectar fins a 2 motors que funcionin de manera independent (2 ports OUT són els 2 pols que necessita el motor). No obstant, en el projecte s’han connectat els motors de la banda dreta del vehicle a una sortida i els de l’esquerra a l’altra, de manera que els motors d’una mateixa banda giren de forma idèntica.

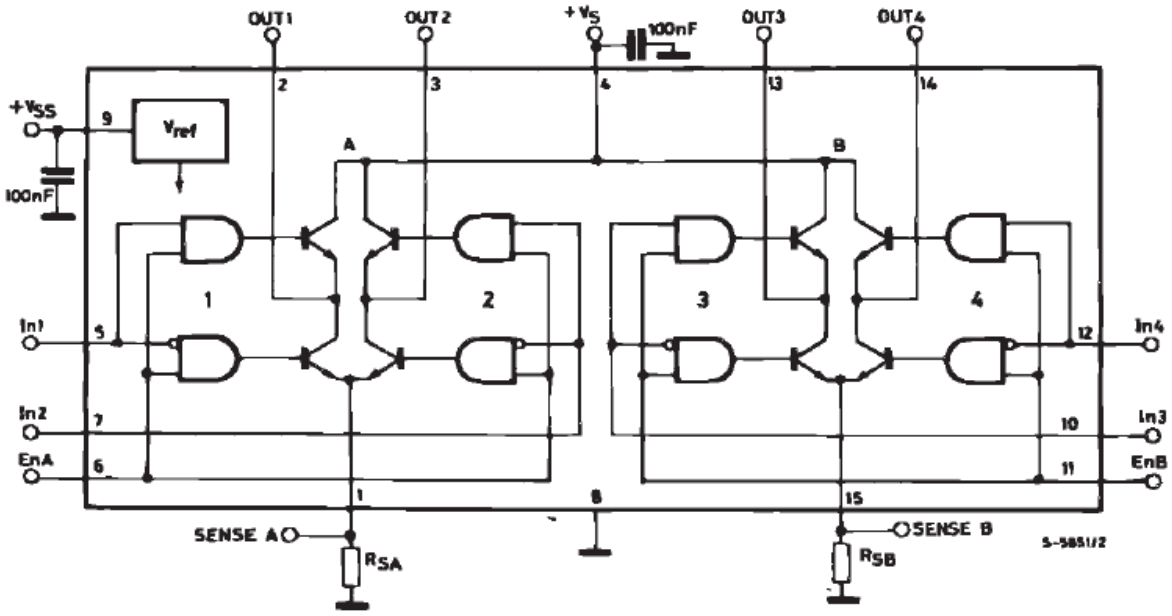


Figura 21: Esquemàtic del xip L298N

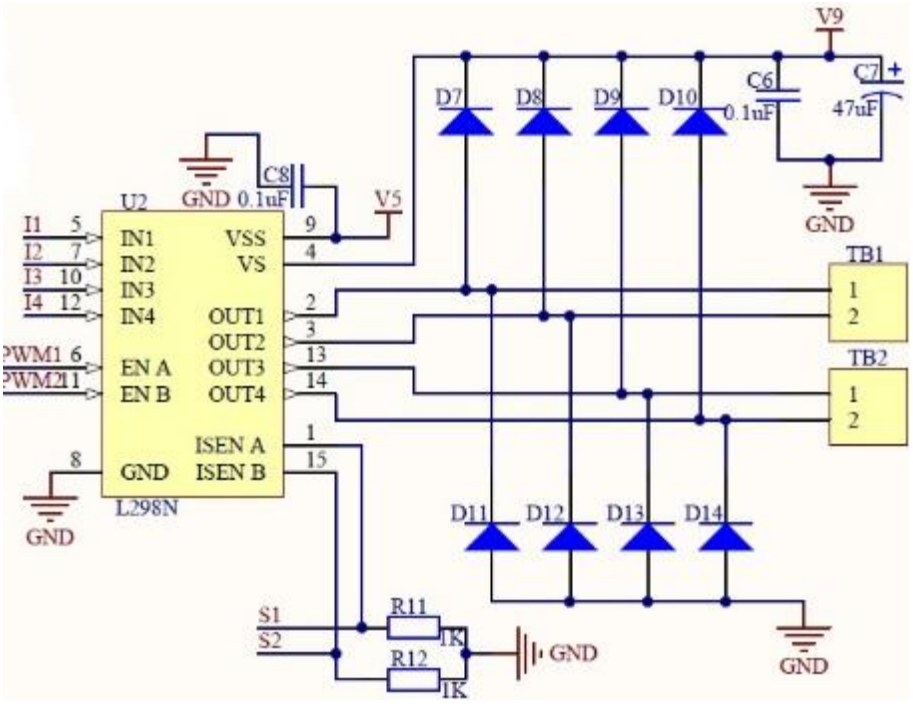


Figura 22: Esquemàtic de la placa que incorpora el xip L298N

També disposa de 4 pins d'entrada anomenats IN, 2 per cada pont. Per cada motor es requereixen també dos pins IN, que serviran per indicar la direcció en la que es desitja fer girar el motor, segons si s'estableix un com a *HIGH* i l'altre com a *LOW*, o a l'inrevés.

Disposa també dels pins d'alimentació. Hi ha dues formes d'alimentar el circuit, depenent de la tensió que necessitin els motors. La primera consisteix en connectar la tensió desitjada per al motor al pin +12 V i deixar el "jumper" que regula la tensió posat (curtcircuitar els dos pins del jumper), i serveix per a alimentar motors d'entre 5 i 12 V. En aquest cas el pin de +5 V ens proporciona una tensió regulada de sortida a aquest valor. La segona manera, consisteix en alimentar pel pin +12 V a la tensió desitjada per al motor, en deshabilitar el "jumper" i en connectar 5 V al pin de +5 V per a alimentar la part lògica del driver. En el cas del projecte, s'ha escollit la primera manera d'alimentar el circuit i els motors, ja que són motors de DC de 5 V.

Per acabar, cal fer menció als dos pins "Enable" (A i B) dels que disposa la placa. Si es volen deshabilitar, només cal curtcircuitar els dos pins mitjançant el "jumper" a través de la placa. Si per contra es vol controlar la velocitat dels motors mitjançant PWM, connectant un pin a una sortida amb PWM de la placa Arduino es podrà controlar la velocitat de gir.

9. Alimentació del sistema

L'objectiu del projecte es que el vehicle sigui capaç de funcionar de manera autònoma. Caldrà alimentar tots els elements del circuit, i que la tensió subministrada a tots ells sigui tal que no superi els màxims permesos pels fabricants ni sigui menor al mínim especificat, ja que aleshores el component en qüestió no funcionaria.

9.1 Càlcul de consums i tensió necessària

Abans de decidir com s'alimentarà el sistema, és necessari detenir-se i calcular el consum total del vehicle. Aquest ens donarà una idea de com subministrar l'energia depenent de les limitacions de cadascuna de les opcions que es veuran a l'apartat 9.2. A més, cal analitzar també les tensions de funcionament de tots els components, i considerar si tots es poden alimentar des de la mateixa font o seran necessàries fonts alternatives o reguladors de tensió. A la taula 5 es mostra la intensitat que consumeix cada component així com també les tensions d'alimentació mínimes i màximes de cadascun.

	Consum	Tensió
Sensor *HC-SR04*	15 mA	5 V
Driver *L298N*	36 mA	5 – 46 V
Arduino Uno R3	65 mA	6,2 – 12 V
Motors DC	100 mA	3 – 7,5 V

Taula 5: Consums i tensions dels diferents mòduls emprats

Com que es tenen 2 sensors ultrasònics i 4 motors, el consum final és de 531 mA. El consum de la placa d'Arduino pot ascendir fins als 80mA en funció de les tasques que es requereixin d'ella. En tot cas, en el pitjor dels casos, el consum seria d'uns 546 mA. Com que s'ha especificat que l'autonomia del sistema ha de ser com a mínim d'1 hora (apartat 4.5), serà necessària una bateria de mínim 546 mAh de capacitat.

Pel que fa a les tensions, es veu que com a mínim caldrà una font que proporcioni 6,2 V.

9.2 Alternatives i elecció final

Una primera alternativa hagués estat alimentar la placa Arduino a través del port USB amb

una bateria externa, que proporciona una tensió d'alimentació regulada de 5 V. A continuació es podria fer servir el pin 5 V per alimentar els sensors i el driver, que al seu torn, alimentaria els motors. No obstant, amb una tensió de 5 V, el driver no treballa amb tota la tensió que podria treballar, i pot perdre una mica d'eficiència.

La segona opció consisteix en alimentar Arduino a través del pin VIN amb una tensió compresa entre 6,2 i 12 V. Aquesta mateixa tensió es pot utilitzar per alimentar el driver i els motors. Finalment, es podria usar la sortida de tensió regulada de 5 V del pin 5V d'Arduino per a alimentar els sensors ultrasònics.

En primer lloc, es troben les bateries de 9 V molt esteses al mercat. El voltatge és apropiat per a alimentar tots els mòduls. Tenen l'avantatge de ser fàcils de trobar i d'utilitzar. A més hi ha disponibles caixes i porta-piles, que inclús alguns incorporen un connector jack igual que el que utilitza Arduino. El principal desavantatge és que disposen de baixa densitat energètica, disposant d'una capacitat típica de 500-600 mAh, fet que no assoleix les especificacions del sistema, o de manera molt limitada. A més, proporcionen una intensitat de corrent màxima molt baixa, al voltant dels 300 mA, només útil per a projectes petits. Finalment, la tensió de 9 V és excessiva per als motors.

Figura 23: Bateria de 9 V i porta-bateria

En segon lloc es troben les piles AA d'1,5 V. Per a alimentar tots els mòduls amb aquestes bateries, caldrien un total de 5 piles connectades en sèrie. Tenen l'avantatge de ser fàcils de trobar. Cinc piles AA convencionals proporcionen 800-1500 mAh, mentre que en el cas de piles AA alcalines la capacitat és de 1700-2800 mAh. La intensitat màxima que es pot obtenir és superior a 1 A, podent arribar fins als 2 A. El preu d'aquestes piles és relativament barat, però al no ser recarregables, a llarg termini resulta no ser econòmic. Ha estat difícil trobar en el mercat encapsulats que permetin la connexió de 5 bateries AA en sèrie. Es va intentar agafar-ne un de 6, i curtcircuitar una de les posicions, soldant un fil de coure entre els extrems de dues piles consecutives. No obstant, degut al recobriment que tenia el ferro de l'encapsulat, l'estany no es soldava bé i es va descartar aquesta opció.

Figura 24: Bateries AA de 1,5 V i porta-piles de 6

Una tercera opció seria emprar bateries recarregables AA de 1,2 V. Es poden utilitzar bateries de NiCd (en desús) o NiMh. Com que el voltatge és una mica inferior a les anteriors, caldria utilitzar-ne 6 per aconseguir 7,2 V. Les piles recarregables AA són fàcils de trobar, però són més cares que les no recarregables. A més, cal afegir-li la necessitat d'un carregador adient, fet que encareix l'alimentació mitjançant aquest mètode. La capacitat d'aquest tipus de bateries és de 500-1000 mAh per les NiCd i 600-2500 mAh per les NiMh. La intensitat màxima és també d'1A. Aquesta opció és viable per al projecte, l'única desavantatge és el preu d'adquisició. El cost d'una pila ronda els 1,05 € i el del carregador uns 8 €.

La quarta opció consisteix en utilitzar dos bateries de liti 18650 de 3,7 V connectades en sèrie. Això proporcionaria una tensió d'alimentació de 7,4 V, que és un voltatge perfecte per alimentar Arduino. Tenen l'avantatge de tenir una alta capacitat de càrrega. Les bateries de primera marca poder arribar a oferir fins a 4800 mAh. La intensitat màxima que poden subministrar és de 10 A, encara que per seguretat s'aconsella no drenar més de 2 - 4 A. Són bateries recarregables, encara que es requereixen uns carregadors especials per a aquestes bateries, el preu dels quals ronda els 7 €. Són una mica més cares que les anteriors, amb un preu d'uns 3 € per bateria. A més, la seva manipulació ha de ser més curosa, ja que un ús incorrecte pot provocar sobreescalfaments i fins i tot incendis. És una altra opció viable per al projecte, si més no la desavantatge principal també radica en el preu.

Figura 25: Bateries de liti 18650 i porta-piles de 2

Així doncs, les dos opcions més viables són la 3a i la 4a. Malgrat que el preu de les piles és una mica més car i a més cal comprar un carregador adient, a la llarga això podria abaratir els costos comparat amb les bateries no recarregables.

Finalment, però, deixant de banda tot l'anterior, es va aconseguir recuperar una bateria d'un antic projecte. És una bateria de vehicles de radio control de 7,2 V i 1200 mAh de capacitat. A més, la bateria permet ser recarregada. Junt amb ella venia un carregador amb transformador incorporat, que permet recarregar-la directament de la xarxa elèctrica. Als bornes de la bateria, se li va soldar un interruptor que permetés deixar d'alimentar el vehicle si així es desitja, evitant desendollar cables. L'única desavantatge és que és una bateria vella, i el pes que afegeix al vehicle es considerable. A més, ocupa més espai que totes les opcions anteriors, fet que provoca una major optimització d'aquest.

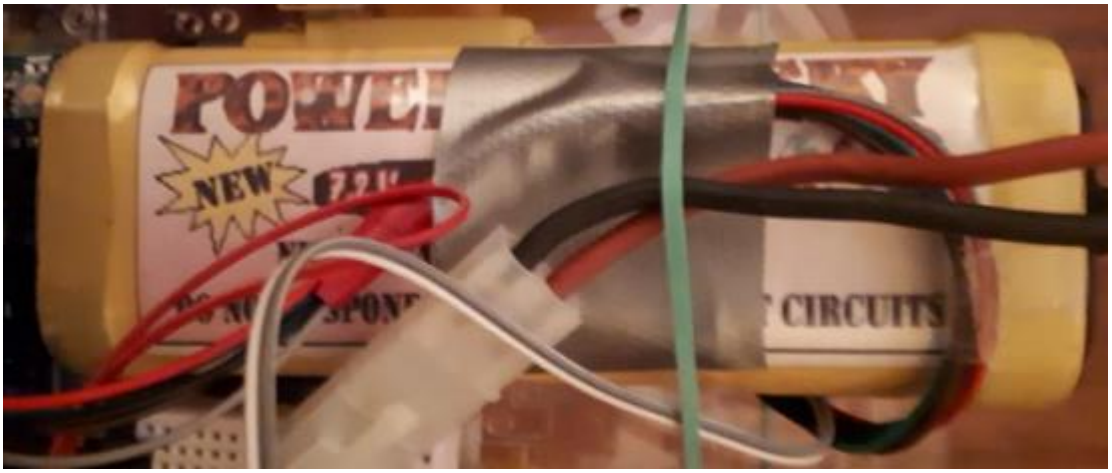


Figura 26: Bateria de RC emprada en el projecte

10. Metodologia

En aquest capítol s'explicaran els passos que s'han seguit per construir l'aplicació i les eines que s'han utilitzat.

El projecte s'ha desenvolupat per fases, de manera seqüencial. Per a cada component del projecte, s'ha escrit un codi en l'entorn de desenvolupament d'Arduino *IDE* i s'ha provat el funcionament.

S'ha començat pel driver controlador de motors *L298N*, fent que controlés la direcció i la velocitat de gir dels dos grups de rodes.

A continuació, s'ha desenvolupat un programa que permeti l'obtenció de les distàncies mitjançant els sensors ultrasònics *HC-SR04*. Primer s'ha mesurat només amb un sensor, i després ja s'ha escrit un codi complet per a la integració de les dos mesures. A mesura que s'avançava en el projecte i es feien proves de conducció, es va veure que de vegades els sensors no proporcionen informacions fiables, i en una segona sofisticació de l'obtenció de les mesures es va procedir a establir un filtre que proporcionés un senyal més fiable i estable en el temps.

Després s'ha escrit el codi complet del programa que integra tant l'obtenció de les distàncies per part dels sensors, com l'elecció de la velocitat dels motors per tal que el vehicle corregeixi la seva trajectòria per situar-se en el punt desitjat. Amb el temps, el programa ha estat sotmès a modificacions importants que seran esmentades més endavant.

Finalment, un cop integrats tots els components del sistema, s'han fet les proves de conducció pertinents i s'han establert els límits del projecte.

11. Configuració del hardware

En aquest capítol s'explicarà com estan connectats entre si els sensors ultrasònics de distància, el driver controlador de motors, la plataforma Arduino, els motors i la bateria externa. Per a cada component s'explicarà quins pins de la placa Arduino s'han fet servir per al seu control i quins muntatges s'han fet per evitar conflictes elèctrics. Per a facilitar les connexions s'ha utilitzat una mini-board que permet connectar diversos punts a una mateixa tensió.

11.1 Connexió dels sensors ultrasònics *HC-SR04*

El mòdul ultrasònic *HC-SR04* consta de 4 pins que s'han de connectar per a què funcioni de manera correcta. El pin VCC i el pin GND serveixen per alimentar el mòdul. Com que és un mòdul que s'alimenta a 5 V, es farà servir el pin 5 V de la placa Arduino per subministrar aquesta tensió. Ja que la placa s'alimenta pel pin VIN a 7,2 V, el regulador de tensió que duu incorporat transforma aquesta tensió a una tensió regulada de 5 V que es subministra pel pin anomenat 5 V.

Aquesta sortida de 5 V s'ha connectat en un punt de la mini-board, on al seu torn s'hi han connectat els dos pins VCC dels dos sensors. Per altra banda, els pins GND s'han connectat al punt de la mini-board on hi ha totes les tensions "massa" o "terra".

Pel que fa als pins Trigger, el del sensor dret s'ha connectar al pin 5 de la placa Arduino i l'esquerra al pin 2. Aquests pins són de tipus digital i permetran enviar el senyal pertinent al sensor, que l'obligui a enviar els 8 polsos de 40 KHz.

Finalment, el pin Echo del sensor de la dreta s'ha connectat al pin 3 de l'Arduino i l'esquerra al pin 4. Són pins també de caràcter digital que permetran detectar quanta estona ha estat el pin Echo a nivell *HIGH* i així poder trobar la distància.

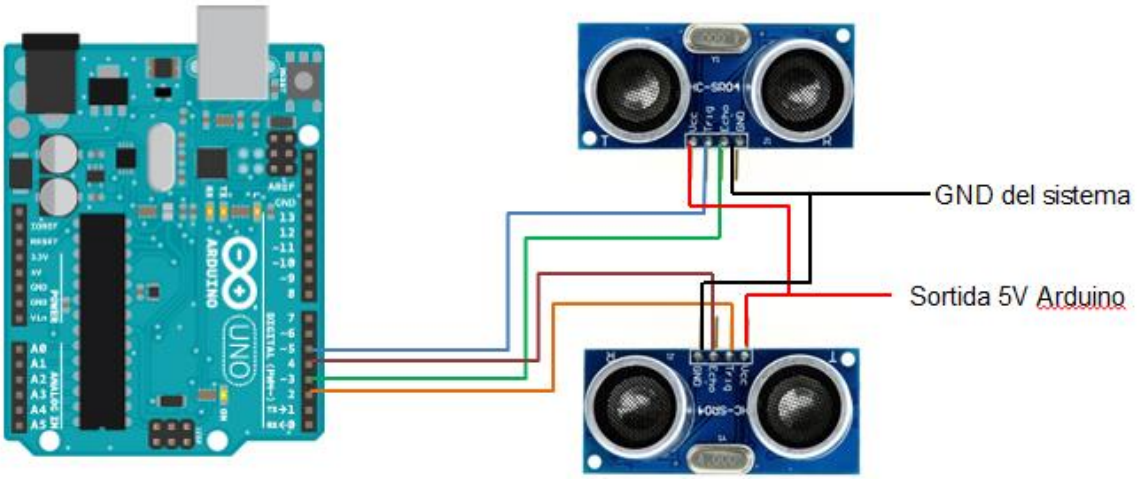


Figura 27: Connexió dels sensors ultrasònics HC-SR04 a l'Arduino i l'alimentació

11.2 Connexió del driver controlador de motors *L298N*

Pel que fa al driver, s'ha connectat al pin +12 de la placa l'alimentació de 7,2 V provinent de la bateria externa i el punt GND al punt de terra global del sistema, ambdues connexions a través de la mini-board. Com que l'alimentació no és superior a 12 V, no cal treure el "jumper" ja que així no és desactiva el regulador de tensió i el pin +12 V de la placa queda connectat al pin +Vss del xip.

Els pins IN1 i IN2, que corresponen als motors de la banda esquerra, s'han connectat als pins 8 i 7 de la plataforma Arduino, respectivament. Els pins IN3 i IN4, corresponents als motors que mouen les rodes de l'esquerra, s'han connectat als pins 13 i 12 de l'Arduino, respectivament. En la posterior configuració d'aquests pins via software, l'un s'establirà a nivell *HIGH* i l'altre a nivell *LOW*. Si es vol invertir la direcció de gir de les rodes, només cal invertir els nivells d'aquests dos pins.

Els pins ENA i ENB s'han connectat als punts de sortida digital 9 i 11 de l'Arduino. És necessari que aquests punts estiguin connectats a una sortida digital amb *PWM*, ja que serà necessari enviar un senyal pseudo-analògic a través d'aquests per establir la velocitat de gir dels motors.

Finalment, als ports OUT1 i OUT2 del driver s'hi ha connectat els bornes dels motors de la banda esquerra. Cal connectar els dos pols negatius a la sortida OUT1 i els dos pols positius a la sortida OUT2. Si s'intercanvia la polaritat, novament els motors giraran en sentit contrari, però això no es cap problema, ja que es podrà canviar posteriorment a través del codi carregat al microcontrolador. Finalment, cal fer el mateix amb els pols dels motors de la dreta, els dos negatius al pin OUT3 i els dos positius al pin OUT4.

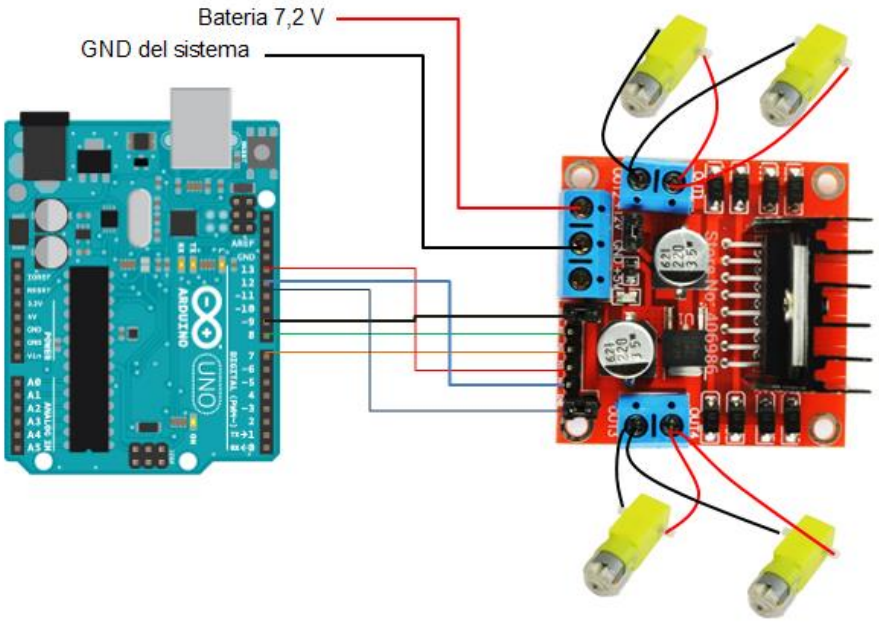


Figura 28:Connexió del driver L298N a l'Arduino i a l'alimentació i dels motors al driver

11.3 Connexió de la placa Arduino

A banda dels pins que ja s'han esmentat en els apartats anteriors, cal alimentar la placa amb una tensió d'entre 6,2 V i 12 V tal i com recomana el fabricant. Com que el regulador de tensió que incorpora Arduino té una caiguda de tensió mínima d'uns 1,2 V, i els mecanismes interns funcionen amb 5 V, alimentar-lo amb una bateria externa de 7,2 V ha estat la manera òptima d'obtenir un resultat segur. Aquesta alimentació requereix connectar el pin VIN al pol positiu de la bateria i el pin GND al pol negatiu. Com que els pols de la bateria també són requerits pel driver controlador, les connexions s'han fet de nou a través de la mini-board. Finalment, del pin 5 V en surt una tensió regulada que serveix d'alimentació per als dos sensors de distància, com ja s'ha esmentat abans.

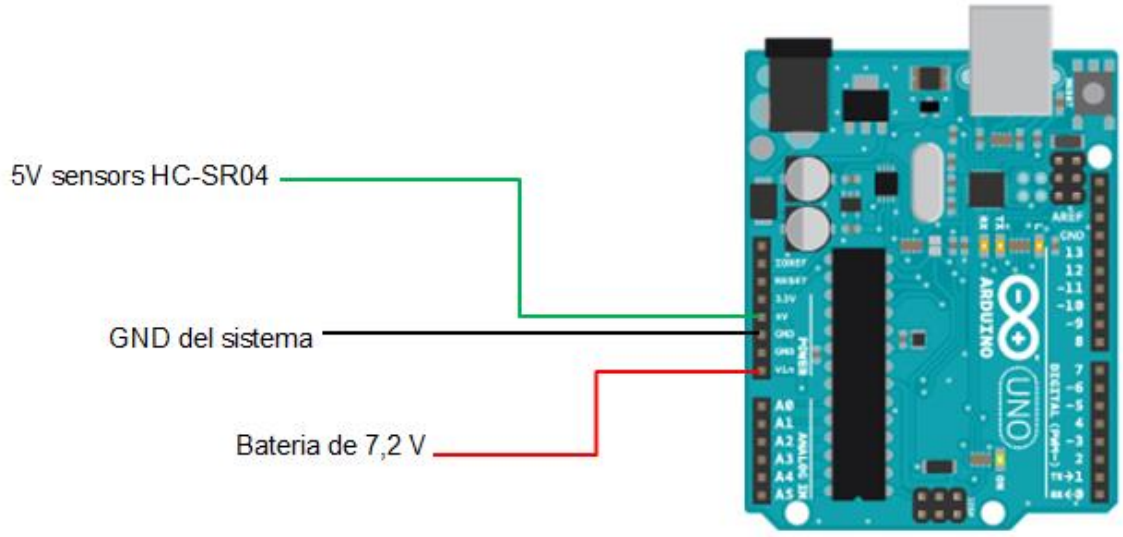


Figura 29: Connexió de l'alimentació de la placa Arduino i sortida del pin 5V

12. Descripció del programa d'Arduino

En aquest apartat s'explicarà l'evolució del programa del microcontrolador de l'aplicació final, així com el programa definitiu que fa funcionar el vehicle de manera òptima. En cada cas es descriuran la funció i les instruccions que desenvolupa la placa d'Arduino i com està estructurat el programa. A l'annex A s'inclou el codi final sencer amb comentaris del programa del microcontrolador.

12.1 Primera versió del programa

Com que els coneixements previs de programació, i en concret en el llenguatge que utilitza Arduino IDE, no eren molt amplis, es va començar fent un programa senzill i intuïtiu. El funcionament del mateix era el descrit a continuació.

Es buscava la distància de cada banda. Per a fer-ho, es cridava a una subrutina que encenia 10 µs el pin Trigger (fet que provoca l'emissió dels polsos ultrasònics), es calculava el temps que el senyal trigava en anar i tornar, es dividia el temps per la meitat (ja que només interessa l'anada o la tornada) i finalment es convertia el temps a centímetres utilitzant el valor mig de la velocitat del so en l'aire.

A continuació, es comparaven les dues mesures i si, per exemple, la distància per la dreta era més petita que la de l'esquerra, s'apagaven les rodes de l'esquerra (posant els dos pins IN1 i IN2 a nivell LOW) i es deixaven girar només les de la dreta, per què així el vehicle virés.

Aquest sistema de funcionament resultava molt rudimentari, ja que els sensors tenen una precisió de 3 mm, i el vehicle girava contínuament d'una banda a l'altra, sense seguir una trajectòria suau. A més, el factor més important dels sensors, com ja s'ha esmentat, és l'angle d'inclinació. Sovint, el gir era tan brusc que el sensor deixava de percebre la paret lateral i el vehicle s'estavellava contra ella.

D'altra banda, es van començar a detectar els primers errors de mesura/programació en els sensors. En primer lloc, si després d'estar mesurant de manera estable, s'afegia una pertorbació momentània (com la detecció d'un objecte molt més proper del mesurat fins llavors), el programa retornava una mesura molt més gran del normal, fins i tot de vegades fora del rang de mesura del que disposa el sensor (4 m). Per a solucionar això es va afegir una distància màxima de detecció de 3 m que s'ha mantingut constant en totes les posteriors versions del codi. En segon lloc, de vegades el programa retornava valors de mesures negatives, també en casos d'aparició de pertorbacions instantànies. Es va veure que el problema venia de la utilització en el codi de nombres decimals o de coma flotant

(*floating point* en anglès, *float* al codi). Quant el programa vol retornar un valor que sobrepassa el que li permeten el seu nombre de bits, reprèn el comptatge des del nombre negatiu més baix que permet representar.

Fins ara les mesures s'obtenien en centímetres, amb un decimal que indicava els mil·límetres. Per a solucionar el problema dels nombres negatius es va decidir eliminar totes les variables que fossin de tipus *float* i passar-les a enteres (*int*). Per a no perdre la precisió mil·limètrica que retorna el sensor, es va multiplicar el nombre per 10 per així obtenir les distàncies en mm.

12.2 Implementació d'un controlador PID

Per resoldre el problema dels canvis continus de direcció i velocitat en les rodes es va procedir a la incorporació d'un controlador PID (Proporcional Integral Derivatiu) a través del software.

Per a definir un controlador PID calen bàsicament 3 elements: una entrada al controlador (anomenada *Input*), una sortida (anomenada *Output*) i un punt de consigna al que es desitja establir el sistema (anomenat *Setpoint*). L'equació bàsica de tot controlador PID és la següent:

$$Output = Kp \cdot e(t) + Ki \cdot \int e(t)\,dt + Kd \cdot \frac{d}{dt}e(t)$$

on: $e = Setpoint - Input$

En altres paraules, un cop calculat l'error, definit com la diferència entre el punt on es vol arribar (*Setpoint*) i el punt on està el sistema actualment (*Input*), es calcula la sortida com una funció d'aquest error.

En primer lloc, una part de la sortida (*Output*) és directament proporcional amb constant *Kp* a aquest error mesurat, obtenint així la part Proporcional del controlador.

En segon lloc, es calcula la integral d'aquest error en el temps i es multiplica per una certa constant *Ki*, obtenint la part Integral. Per obtenir la integral en el codi del programa, com que l'error són nombres constants per cada instant de temps, el que s'ha fet és sumar l'error actual a la suma acumulada dels errors, ja que la integral d'una suma és la suma d'integrals.

Finalment, es calcula la derivada de l'error en el temps i es multiplica per una constant *Kd*, obtenint-se així la part Derivativa del controlador. La implementació d'aquesta derivada a través del codi és una feina més costosa i s'explicarà més endavant.

Així doncs, en definitiva, un controlador PID calcula l'error entre l'entrada al sistema (*Input*) i

el punt on es vol arribar (*Setpoint*). Després intenta minimitzar l'error ajustant la sortida (*Output*).

El controlador PID ha estat implementat seguint una llibreria d'Arduino feta per Brett Bauregard encarada a aquest fi. Bona part del mèrit del funcionament del controlador PID ha estat gràcies a aquesta llibreria i a l'extensa documentació sobre ella que el creador ha posat a disposició dels usuaris.

12.2.1 Característiques principals de la llibreria PID

Una llibreria d'Arduino són traces de codi escrites per tercers i que s'utilitzen en l'sketch. Aquest fet facilita molt la programació i fa que el programa resulti més senzill de realitzar i d'entendre. La llibreria PID va ser dissenyada amb la finalitat de poder establir un controlador d'aquest tipus i alhora fer-lo robust davant de problemes comuns que sorgeixen a l'hora d'implementar-lo. A continuació es descriuen les característiques principals de la llibreria que permeten obtenir un controlador PID que funcioni de manera òptima.

El controlador PID (d'ara en endavant PID) està dissenyat per a ser cridat pel programa en qualsevol instant. Això provoca dos problemes.

- No s'obté un comportament consistent del PID, ja que de vegades és crida freqüentment i de vegades no.
- Cal aplicar càlculs matemàtics extra per a calcular la part derivativa i la integral, ja que les dues depenen del canvi de temps.

Per a solucionar-ho, la llibreria s'assegura que el PID és cridat pel programa en un interval regular i constant, de manera que la funció principal del PID, anomenada "Compute()", és cridada cada cicle del programa. Basada en un temps de mostreig predeterminat, el PID decideix si ha de fer el càlcul o retornar immediatament.

Un cop està assegurat que el PID és avaluat en un interval constant, els càlculs de la part derivativa i integral se simplifiquen.

El segon problema consisteix en un fet anomenat "cop derivatiu" per l'autor de la llibreria. La modificació del codi per a solucionar això, farà canviar una mica el terme derivatiu del controlador. L'objectiu és aconseguir eliminar aquest fenomen.

La figura 30 il·lustra el problema. Ja que l'error és la resta entre el *Setpoint* i l'*Input* , qualsevol canvi en el *Setpoint* provoca un canvi instantani en l'error. La derivada d'aquest canvi és infinita (a la pràctica, com que *dt* no és realment 0, simplement acaba provocant un nombre molt gran). Aquest nombre s'introdueix en l'equació del PID, fet que provoca un pic

no desitjat en l'*Output*.

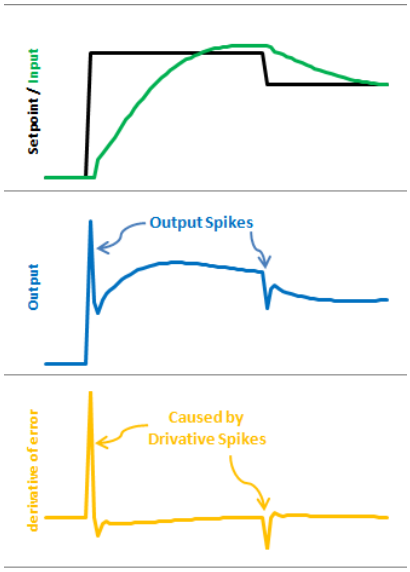


Figura 30: Problema del "derivative kick" que provoca un pic en la sortida

Hi ha una manera de desfer-se d'aquest problema. Ja que:

$$\frac{dError}{dt} = \frac{dSetpoint}{dt} - \frac{dInput}{dt}$$

Quan el *Setpoint* és constant:

$$\frac{dError}{dt} = -\frac{dInput}{dt}$$

Resulta que la derivada de l'error és igual a menys la derivada de l'*Input*, excepte quan el *Setpoint* està variant. Això resulta ser una solució perfecta i a més proporciona una manera més senzilla de calcular la part derivativa, i el resultat es mostra en la gràfica de sota.

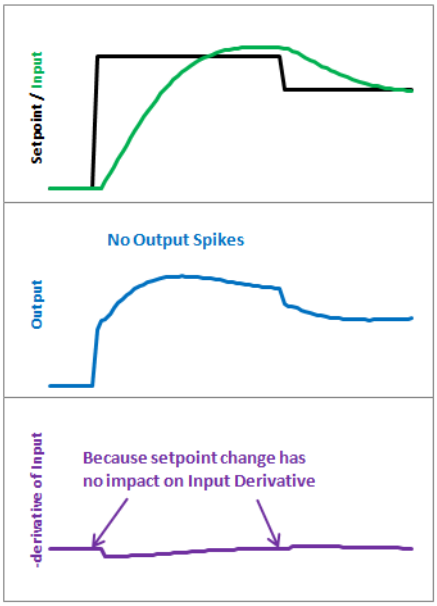
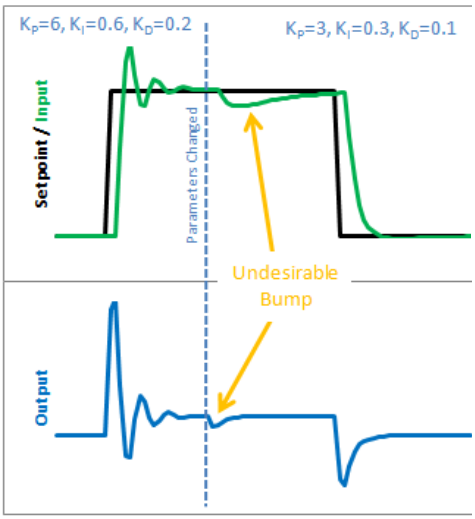


Figura 31:Sortida del sistema un cop solucionat el fenomen del "derivative kick"

Un altre problema que la llibreria resol de manera eficaç és el fet de poder modificar els valors dels paràmetres del PID (*Kp*, *Ki* i *Kd*) mentre el sistema està funcionant, fet que és gairebé obligatori en qualsevol algoritme que implementi un PID de manera eficient i robusta.

En les proves fetes pel dissenyador de la llibreria, es va veure que el PID actuava de manera errònia si s'intentava canviar els paràmetres mentre estava en marxa. Al gràfic de sota es mostra aquest fet.



Output is halved

	Output	= kp * error +	ki * errSum	- kd * dInput
Just Before	0.98	6 -0.01	0.6 1.73	-0.2 0.02
Just After	0.49	3 -0.01	0.3 1.72	-0.1 -0.01

Because the Integral Term suddenly gets ha

Figura 32:Valor de la sortida en canviar els paràmetres del controlador

Es pot culpar directament d'aquesta pertorbació (*Undesirable Bump* al dibuix) al terme integral del PID. És l'únic terme que canvia dràsticament quan els paràmetres canvien, degut a la interpretació de la integral per part del PID:

$$Ki \int e\,dt \approx Ki_n[e_n + e_{n-1} + \cdots]$$

Aquesta interpretació funciona bé fins que el valor de *Ki* canvia. Llavors, de sobte, es multiplica el nou valor de *Ki* per tota la suma de l'error que s'ha anat acumulant. Això no és el que es pretén, ja que només es vol que el canvi afecti als paràmetres en endavant.

La solució és bastant senzilla però eficaç. En comptes de tenir *Ki* fora de la integral, s'introdueix a dins. Aleshores el que es fa és agafar l'error i multiplicar-lo pel valor de *Ki* en aquell moment, i a continuació anar emmagatzemant la suma d'això.

$$\int Ki \cdot e\,dt \approx Ki_n e_n + Ki_{n-1} e_{n-1} + \cdots$$

Quan canvia la *Ki*, no es produeix cap pertorbació ja que els valors anteriors de *Ki* estan emmagatzemats. Es produeix així una transició suau sense càlculs matemàtics addicionals. El resultat es mostra a continuació.

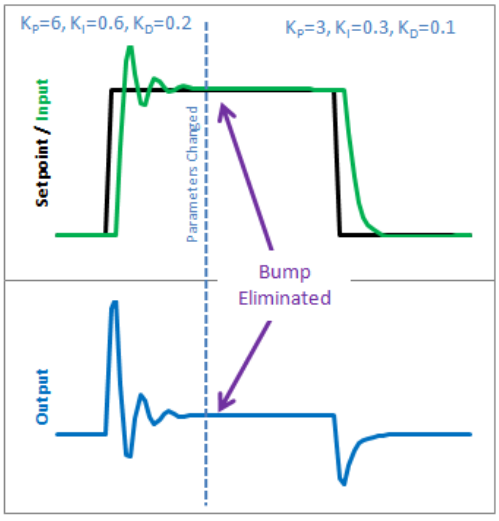


Figura 33: Eliminació del pic en canviar els paràmetres del controlador

El següent aspecte té un nom que vindria a ser "Finalització del reinici" (*Reset Windup* en anglès). Aquest fet és una trampa que probablement aparegui en la majoria d'usuaris de controladors PID, ja siguin de nivell avançat o no. Succeeix quan el PID pensa que és capaç

de fer quelcom que realment no pot fer. Per exemple, els valors de *PWM* de la placa Arduino van des de 0 fins 255. Per defecte, el PID no és conscient d'això. Si creu que un valor superior a 255 funcionarà per dur el sistema al punt desitjat, provarà aquests valors a l'espera d'obtenir el que necessita. Però com que el sistema es troba restringit a 255, simplement seguirà intentant nombres cada cop més alts sense arribar enlloc.

El problema es presenta en forma de retards (*lag,* en anglès). En la figura 34 es pot veure com l'*Output* és molt més gran que el límit extern del sistema. Quan el *Setpoint* varia, l'*Output* ha de disminuir abans de creuar de nou la línia del límit.

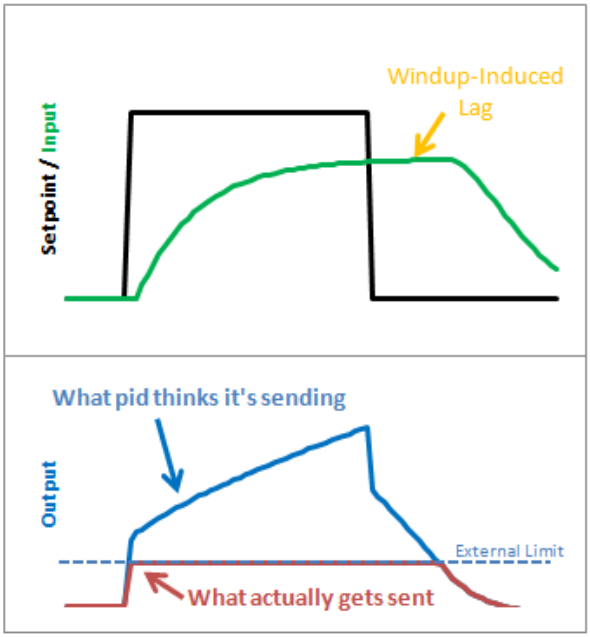


Figura 34: Retard i Output per sobre del que permet el hardware

Hi ha diverses formes de mitigar el retard, però la que va escollir l'autor de la llibreria és la següent: especificar al PID quins són els límits de la sortida (*Output*). Un cop s'arriba al límit, el PID deixa de sumar (integrar). Ja que l'output no es dispara, s'obté una resposta immediata quan el *Setpoint* decau en un rang on el controlador pot actuar.

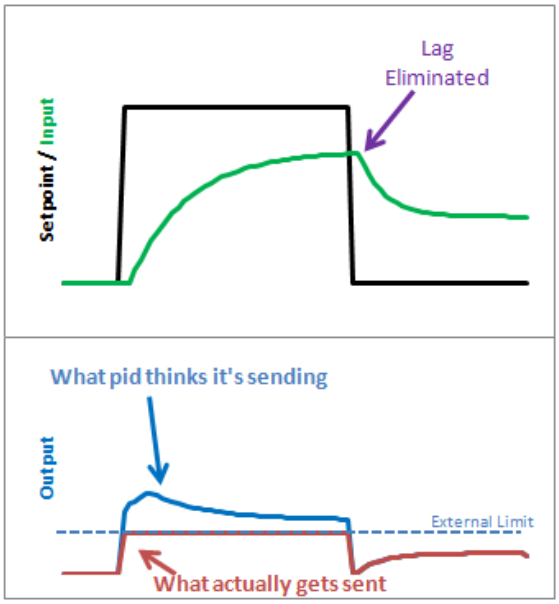


Figura 35: Eliminació del retard i reducció de l'Output gràcies a la constant Ki

No obstant, en el gràfic superior s'observa que malgrat ja no apareix un retard en la resposta, el sistema encara no és del tot òptim. Hi ha encara una petita diferència entre el que el PID creu que està enviant, i el que realment s'està enviant als actuadors. Aquest fet és degut al terme proporcional i al derivatiu.

Malgrat que el terme integral ha estat restringit de manera correcta, la part proporcional i la derivativa encara afegeixen la seva petita aportació, proporcionant un resultat més gran que els límits de sortida. Per a solucionar-ho la llibreria compta amb una funció anomenada "*SetOutputLimits()*" que restringeix els valors de sortida a un màxim i un mínim que poden ser establerts per l'usuari, en funció de les característiques del sistema a controlar.

La pregunta que sorgeix a continuació és: si es restringeix la sortida de totes maneres, per a què fer-ho en el terme integral de manera separada? Si tot el que es fes fos limitar la sortida, el terme integral aniria de nou creixent cada cop més. Encara que l'*Output* es veiés bé durant la implementació, es veuria el retard quan el sistema canviés el *Setpoint*.

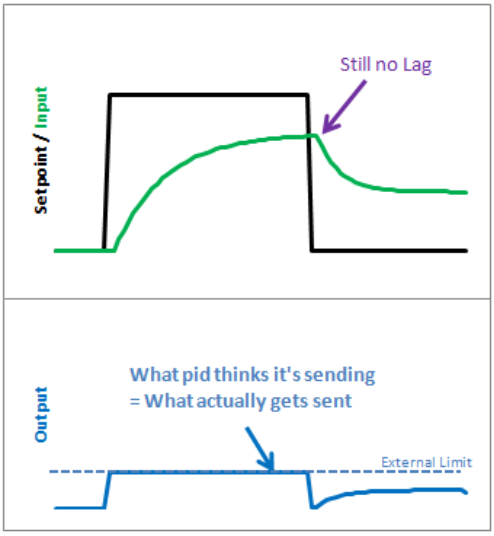


Figura 36: Eliminació del retard i sincronització del l'Ouput del software i el hardware

D'altra banda, per molt útil que sigui tenir un controlador PID implementat, de vegades no interessa el que ell vulgui aportar. Si en un moment donat es vol forçar la sortida a un cert valor concret, certament es podria fer en la rutina del software, substituint la variable *Output* pel valor que l'usuari desitgi.

No obstant, a la pràctica, això resulta ser una mala idea. El controlador PID quedarà confós, ja que continua movent l'*Output* però observa que el sistema no varia, i llavors intenta moure'l cada vegada més. El resultat serà que quan es deixi de sobreescriure el valor de la sortida i utilitzar el del PID de nou, s'obtindrà un canvi gran i immediat en el valor de *Output*.

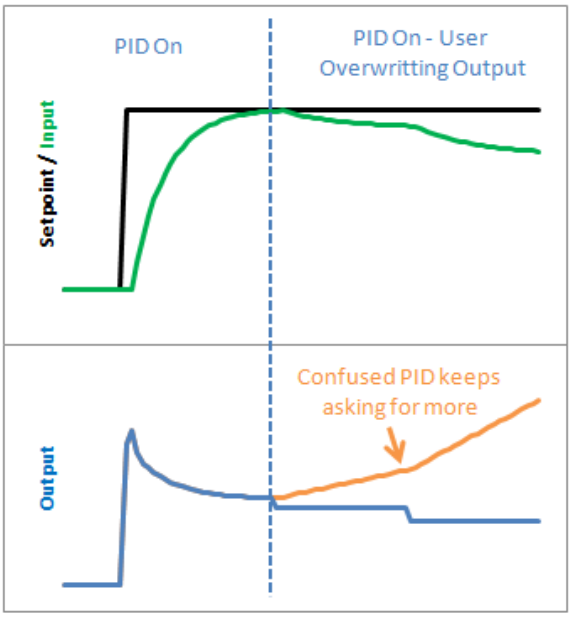


Figura 37: Sobre escriptura de l'Output i valor del PID

La solució al problema consisteix en tenir mitjans per encendre i apagar el controlador. Els termes comuns per a aquests estats són “Automàtic” (el PID ajusta de manera automàtica la sortida) i “Manual” (l’usuari ajusta el valor a mà). Si s’està actuant en mode “Manual”, immediatament el programa deixa d’utilitzar la funció principal “Compute()” (veure apartat 12.2.2) sense ajustar la sortida ni cap variable interna.

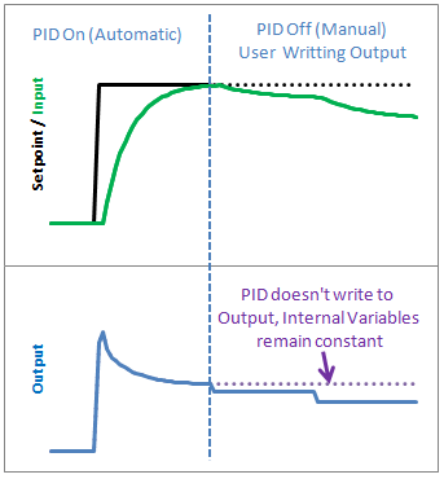


Figura 38: Output quan el PID es troba apagat (Manual)

És cert que es podria haver aconseguit un efecte similar simplement no cridant a la funció “Compute()” de la rutina principal, però aquesta funció manté els mecanismes del PID continguts. A més, d’aquesta manera es pot tenir una traçabilitat de en quin mode s’està treballant, i ens permet saber quan canviar el mode. Això condueix al següent aspecte.

Quan s’apaga el PID i s’encén de nou, succeeix el que es mostra a la figura 39.

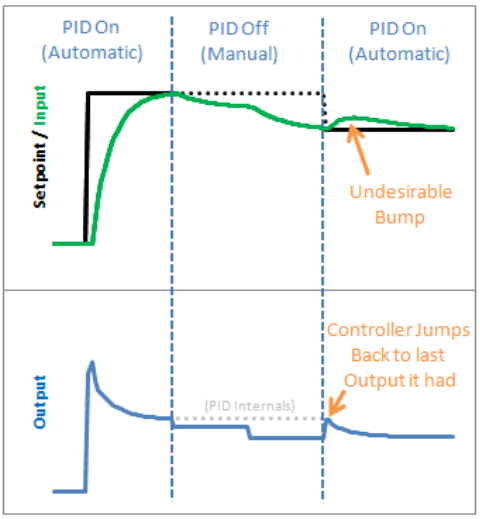


Figura 39: Valor de l’Input quan el PID es torna a encendre

Efectivament, com s'aprecia en el gràfic, el PID torna enrere a l'últim valor de sortida que havia enviat, i continua ajustant des d'aquest punt. Això resulta en una pertorbació en l'*Input* que seria convenient no tenir.

Aquest problema és força senzill de solucionar. Ja que se sap quan s'encén (canvi de "Manual" a "Automàtic"), cal inicialitzar les coses de manera que s'obtingui una transició suau. Això significa suavitzar les 2 variables guardades que treballen (Terme integral i darrer *Input*), per prevenir l'*Output* de salts. Es crea una subrutina per inicialitzar el PID on el darrer *Input* emmagatzemat passa a ser l'*Input* actual (per prevenir pics en la part derivativa) i el terme integral passa a ser l'*Output*. La part proporcional del controlador no depèn de cap informació del passat, així que no requereix cap inicialització. El resultat es mostra en la figura 40.

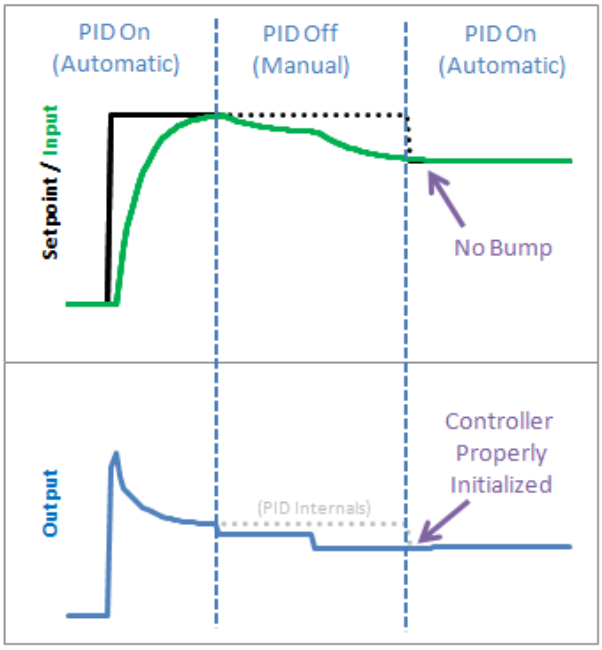


Figura 40: Eliminació dels problemes a l'apagar i tornar a encendre el PID

Ja per acabar, l'última qüestió que resol la llibreria de manera òptima és la direcció en la que treballa el PID. Entre les diferents plantes que es controlen d'aquesta manera, n'hi ha unes en les que un increment en l'Output provoca un increment en l'Input, i d'altres que són inversament proporcionals. El que es fa per resoldre aquest aspecte es mantenir les funcions de la mateixa manera, canviant els signes de les constants *Kp*, *Ki* i *Kd*.

Totes aquestes modificacions fan que aquest algoritme implementi un controlador PID molt robust amb la possibilitat d'interactuar-hi de manera relativament senzilla. De nou cal fer menció a l'autor de la llibreria, Brett Bauregard, que és qui es va adonar de tots aquests problemes i els hi donà solució.

12.2.2 Funcions principals de la llibreria PID

A continuació s'esmentaran les funcions principals de la llibreria, les variables que utilitzen, i els valors que retornen.

PID()

Crea un controlador PID associat a un *Input*, *Output* i *Setpoint* especificats. No retorna res, simplement crea. La sintaxi de creació és la següent: *PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction).*

Totes les variables que necessita són de tipus *double*. Les variables d'aquest tipus són nombres *float* amb el doble de precisió. No obstant, en les placa Arduino Uno, aquesta variable ocupa 4 bytes, i té la mateixa precisió que una variable de tipus *float*. L'única diferència es percep si s'utilitza una Arduino Due, on aquestes variables tenen 8 bytes (64 bits) de precisió.

Compute()

Conté l'algoritme que fa funcionar el controlador PID. La funció ha de ser cridada una vegada cada *loop().* El *loop()* és la funció principal del programa que s'executa repetidament, en forma de bucle. La majoria de les vegades, la funció només retornarà sense fer res (degut al temps de mostreig del qual s'ha parlat anteriorment). A la freqüència especificada per la funció *SetSampleTime()* calcularà un nou *Output*.

La funció no requereix cap variable d'entrada, només és necessari cridar-la. Retorna *True* quan es calcula l'*Output* i *False* quan no s'ha fet res.

SetMode()

Especifica si el PID ha d'estar encès (mode *Automatic*) o apagat (mode *Manual*). El PID està en mode apagat per defecte quan es crea. La funció no retorna res.

La sintaxi és *SetMode(mode).* Cal especificar quin es el mode en majúscules.

SetOutputLimits()

Com ja s'ha esmentat abans, aquesta funció retorna un *Output* contingut en un rang de valors especificat en aquesta funció.

Les variables d'entrada de la funció són el valor mínim i el màxim de sortida. Cal que siguin variables de tipus *double*. No retorna res.

SetTunings()

Els paràmetres del PID (*Kp*, *Ki* i *Kd*) dicten el comportament dinàmic del controlador. Si oscil·larà, si serà ràpid o lent, etc. Al crear el controlador, s'especifiquen uns valors inicials per a aquestes constants. Per la majoria d'usuaris això és suficient. No obstant, hi ha vegades en que els paràmetres necessiten ser canviats mentre el programa corre. Per a això es pot fer servir aquesta funció. Les variables d'entrada són els 3 nous valors de les constants, totes en format *double*.

SetSampleTime()

La funció determina cada quan s'avalua l'algoritme del controlador PID. Per defecte és 200 ms. La variable d'entrada que cal especificar-li a la funció és el temps en ms, i cal que sigui una variable entera (de tipus *int*). No retorna res.

SetControllerDirection()

Si l'Input està per sobre del *Setpoint*, l'*Output* ha de disminuir o augmentar? Depenent de quines coses estiguin connectades al PID, una de les dues pot ser certa. En un cotxe, l'*Ouput* hauria de davallar per reduir la velocitat. En canvi, per a un refrigerador, l'*Output* (refredament) ha de ser incrementat per reduir la temperatura. Aquesta funció especifica a quin tipus de procés està connectat el PID. Aquesta informació també s'especifica quan es crea el PID. La variable d'entrada pot ser DIRECT (com un cotxe) o REVERSE (com un refrigerador).

12.2.3 Implementació del controlador PID al vehicle

Una vegada s'ha entès com funciona la llibreria PID i les seves limitacions, cal decidir com s'aplica en el cas concret del projecte. En la majoria d'aplicacions que incorporen un controlador PID, l'*Input* es llegeix directament de la variable que es vol controlar en l'*Output*. Per exemple, si es vol controlar la lluminositat d'un led, l'Input vindrà donat per un sensor que indiqui el nivell de lluminositat, i l'*Output* serà un valor de lluminositat que serà enviat al led per dur-lo al nivell desitjat en el *Setpoint*.

En canvi, en cas d'aquest projecte, l'entrada i la sortida, són valors diferents i que s'obtenen o s'envien a components diferents. En concret, l'*Input* provindrà de les lectures dels sensors *HC-SR04* i l'*Output* serà la velocitat a la que han de girar els motors per aconseguir situar el vehicle en el centre de la trajectòria. Com que són variables de magnituds diferents, serà una mica més entretingut d'ajustar el valor de les tres constants del controlador (*Kp*, *Ki* i *Kd*).

Per altra banda, es tenen dos *Inputs* (la mesura de cada sensor) i dos *Outputs* (els motors

de la banda dreta i els de la banda esquerra), i es vol implementar només un únic controlador PID. La manera òptima de solucionar aquest fet ha estat la següent.

En primer lloc es prenen les mesures de les distàncies a cada costat. En segon lloc, es calcula la diferència d'aquests valors (dreta – esquerra). Es busca que el *Setpoint* del sistema sigui igual a 0, d'aquesta manera la diferència serà nul·la i el vehicle intentarà situar-se en el centre de la trajectòria. Aquesta resta constarà com a *Input* del controlador.

Pel que fa al problema de la velocitat dels motors, el que s'ha fet és establir, d'entrada, una velocitat mitja a la que es mou el vehicle. Una vegada es té calculat l'*Output* del PID, es suma aquest valor als motors de la dreta i es resta als de l'esquerra.

D'aquesta manera es tenen totes les variables d'entrada i sortida integrades en un únic controlador, fet que simplifica l'execució del programa i també fa més senzilla la tasca d'ajust dels paràmetres del controlador.

12.3 Programa principal

En aquest apartat s'explicarà com funciona el programa definitiu que s'executa en la placa. Es descriuran la funció i les instruccions que desenvolupa i com està estructurat el programa.

12.3.1 Funció

El programa principal és aquell que executa la placa Arduino quan vol utilitzar l'aplicació final. La seva funció és fer que el microcontrolador es comuniqui amb els sensors ultrasònics, amb el driver i aquest amb els motors, tot per tal de complir les especificacions del projecte.

El microcontrolador ha de fer un conjunt de tasques en el menor temps possible: obtenir les distàncies dels dos sensors, fer el càlcul amb el controlador PID i aplicar la sortida d'aquest al driver que fa moure als motors.

Per fer totes aquestes funcions, el microcontrolador ha de fer un conjunt de càlculs i operacions d'una manera estructurada en el temps. L'ordre o estructura d'aquestes operacions es descriu a l'apartat 12.3.2.

12.3.2 Estructura i funcionament

Per entendre el funcionament del programa, és convenient tenir en compte les variables que s'han utilitzat i la seva funció (Taula 6).

Variable	Funció
velocitat_mitja	Valor de velocitat que els motors de cada costat tenen en cas d'estar el vehicle completament al centre de la trajectòria
diferencia_distancia	Variable que inclou la resta entre la distància dreta i esquerra, que servirà com a *Input* del controlador PID
fdistR	Emmagatzema el valor de la distància final de la dreta
fdistL	Emmagatzema el valor de la distància final de l'esquerra
diferencia_velocitat	Variable directament relacionada amb l'*Ouput* del PID que servirà per aplicar una diferència de velocitat als motors
llista_dreta	Variable que conté emmagatzemats els valors de les últimes 5 distàncies mesurades pel sensor de la dreta
llista_esquerra	Variable que conté emmagatzemats els valors de les últimes 5 distàncies mesurades pel sensor de la dreta
myPID	Inclou el controlador PID
sonarR	Sensor *HC-SR04* dret
sonarL	Sensor *HC-SR04* esquerra
limit_inferior	Límit inferior de l'*Ouput* del PID
limit_superior	Límit superior de l'*Output* del PID
distanciaR	Valor en mm mesurat pel sensor de la dreta en el *loop* actual del programa
distanciaL	Valor en mm mesurat pel sensor de l'esquerra en el *loop* actual del programa

pwmRight	Valor de PWM que s'envia al driver L298N per als motors del costat esquerra del vehicle
pwmLeft	Valor de PWM que s'envia al driver L298N per als motors del costat esquerra del vehicle
Input	Variable d'entrada al controlador PID
Setpoint	Variable del PID que indica el valor al qual es vol portar el sistema
Output	Variable de sortida del PID
Kp	Valor la constant de proporcionalitat del controlador PID
Ki	Valor de la constant d'integració del controlador PID
Kd	Valor de la constant de derivació del controlador PID

Taula 6: Definició de les variables del programa definitiu

En el moment d'iniciar-se, el programa comença a fer una sèrie d'operacions. En primer lloc s'inclouen 3 llibreries que faciliten l'execució del programa i simplifiquen el codi.

```
#include <Array.h>
#include <PID_v1.h>
#include <NewPing.h>
```

La llibreria *Array* permet crear conjunts de valors i fer operacions amb ells. En aquest cas s'ha utilitzat per crear els conjunts de les últimes 5 mesures dels sensors. La llibreria *NewPing* és de gran utilitat a l'hora de manejar els sensors ultrasònics. Permet obtenir les distàncies en diversos formats, obtenir el temps en µs que ha trigat el senyal ultrasònic en anar i tornar, i fer càlculs sobre aquestes dades abans de ser lliurades, i a més inclou un codi que aconsegueix eliminar alguns errors en les mesures. La llibreria PID_v1 és la llibreria PID que s'ha explicat en l'apartat 12.2.1.

A continuació es defineixen els pins d'Arduino als quals anirà connectat cada un dels elements del hardware. Entre comentaris s'ha especificat el color del cable per a facilitar el muntatge.

```
//MotorA
int IN1 = 8; //gris
int IN2 = 7; //marron
int ENA = 9; //rodes esquerra // negre
//MotorB
int IN3 = 13; //blau
int IN4 = 12; //vermell
int ENB = 11; //rodes dreta // verd
//Sensor Ultraso Dreta
int TrigR = 5; //Cable marron
int EchoR = 3; //Cable vermell
//Sensor Ultaso Esquerra
int TrigL = 2; //Cable Lila
int EchoL= 4; //Cable taronja
```

A continuació s'inicialitzen un conjunt de variables de les que apareixen a la Taula 6.

```
double Setpoint, Input, Output;
double Kp=3.7, Ki=1.5 , Kd=0.3;
int velocitat_mitja = 125;
int diferencia_distancia;
unsigned long fdistR=0, fdistL=0;
int diferencia_velocitat;

const byte size = 5;

//creem llista dreta
int raw_dreta_array[size]={0,0,0,0,0};
Array<int> llista_dreta = Array<int>(raw_dreta_array,size);

//creem llista esquerra
int raw_esquerra_array[size]={0,0,0,0,0};
Array<int> llista_esquerra = Array<int>(raw_esquerra_array,size);

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

NewPing sonarR(TrigR,EchoR,300);
NewPing sonarL(TrigL,EchoL,300);
```

Una vegada s'ha inicialitzat tots els elements de software, és hora de que el programa d'Arduino es connecti amb el món real i iniciï els diversos components.

En primer lloc es configura el Baud Rate, que és el rati al qual la informació es transmesa en un canal de comunicació. És el nombre de bits que són transmesos per segon. Arduino accepta valors de 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 i 115200. El valor escollit ha estat de 9600. És el que ve per defecte amb el programa i resulta més que suficient per a l'aplicació que es vol realitzar.

```
Serial.begin(9600);
```

En segon lloc s'indica quins són els pins que seran d'entrada d'informació a la placa d'Arduino (INPUT) i els que seran de sortida (OUTPUT). També es configura el sentit de gir dels motors especificant els nivells *HIGH* i *LOW* als seus pols.

```
pinMode(TrigR, OUTPUT);
pinMode(EchoR, INPUT);
pinMode(TrigL, OUTPUT);
pinMode(EchoL, INPUT);
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
pinMode(ENA, OUTPUT);
pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);
pinMode(ENB, OUTPUT);

digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH);
digitalWrite(IN4, LOW);
```

A continuació s'inicia el controlador PID i es deixa un temps per a què tot els elements es col·loquin al seu lloc, i també perquè a l'hora de fer funcionar el vehicle hi hagi temps per situar-lo en el punt d'inici.

```
Setpoint = 0;
//iniciem el PID
myPID.SetMode(AUTOMATIC);
double limit_inferior=-velocitat_mitja;
double limit_superior=velocitat_mitja;
myPID.SetOutputLimits(limit_inferior, limit_superior);

delay(4000); //deixem 4s perquè s'inicii i es coloqui tot
```

Un cop inicialitzats tots els processos, es comença a fer córrer el bucle principal del programa (*loop*), que s'anirà repetint de manera cíclica.

Primerament es mesura la distància per cada banda. La funció *ping()* de la llibreria NewPing retorna el temps en µs que el senyal ultrasònic ha trigat en anar i tornar fins a l'objecte més proper. A continuació es divideix entre 58,4. Aquest valor és fruit de la multiplicació de 2 (només interessa el temps que es triga en anar) per 29,2 (valor fruit de la conversió de µs a distància segons la velocitat del so). Es multiplica per 10 per passar-ho tot a mm. Finalment, com que la variable es de tipus *int*, Arduino la trunca, i es perd precisió. Com que només interessa la precisió fins als mm, per arrodonir a la unitat més propera se suma 0,5 al valor

obtingut. És a dir, els valors on la dècima sigui major a 5, quedaran establerts al valor de la unitat superior gràcies a la suma. Els valors on la dècima sigui menor a 5 quedaran truncats a la unitat sense ser afectats pel valor afegit de la suma.

Un cop calculades les distàncies, s'afegeixen a la llista que inclou els últims 5 valors, seguint el criteri *FIFO* (*First in First Out*, en anglès). D'aquesta llista se'n calcula la mediana. Amb això es pretén eliminar una part del soroll que comporta tota mesura de dades del món real. Si de sobte apareix una dada amb un valor extremadament alt o baix, al fer la mediana aquest valor no es tindrà en compte, ja que segurament quedarà mitigat pels valors del seu costat. Si es fa servir la mitjana, el valor final de la distància queda afectat per aquestes variacions momentànies que es sap que són errònies, i això no interessa.

```
int distanciaR = ((sonarR.ping()*10)/58.4)+0.5;
int distanciaL = ((sonarL.ping()*10)/58.4)+0.5;

//afegim a la llista dreta
llista_dreta[0]=llista_dreta[1];
llista_dreta[1]=llista_dreta[2];
llista_dreta[2]=llista_dreta[3];
llista_dreta[3]=llista_dreta[4];
llista_dreta[4]=distanciaR;

//afegim a la llista esquerra
llista_esquerra[0]=llista_esquerra[1];
llista_esquerra[1]=llista_esquerra[2];
llista_esquerra[2]=llista_esquerra[3];
llista_esquerra[3]=llista_esquerra[4];
llista_esquerra[4]=distanciaL;

fdistR=median(llista_dreta[0],llista_dreta[1],llista_dreta[2],llista_dreta[3],llista_dreta[4]);
fdistL=median(llista_esquerra[0],llista_esquerra[1],llista_esquerra[2],llista_esquerra[3],llista_esquerra[4]);
```

A continuació, es calcula l'*Input* del controlador PID com la diferència entre la distància dreta i l'esquerra. Aleshores el PID fa el càlcul tal i com s'ha esmentat en l'apartat 12.2.2 i s'estableix l'*Output* com a la diferència de velocitat que s'haurà d'aplicar als motors.

```
diferencia_distancia= fdistR-fdistL;
Input = diferencia_distancia;

myPID.Compute(); //funció principal del PID. Calculem Output

diferencia_velocitat = Output;


int pwmRight = velocitat_mitja + diferencia_velocitat;
int pwmLeft = velocitat_mitja - diferencia_velocitat;
```

Per si hi hagués hagut algun problema a l'hora de calcular ja sigui en les distàncies o en el PID, s'afegeix un mur de contenció final per als valors de PWM. Teòricament la funció *SetOutputLimits()* de la llibreria PID ja limita aquest fet, però s'ha cregut convenient afegir

aquesta part per als casos on hi pugui haver errors o per si el PID es trobés en estat *Manual* (Off).

```
if(pwmRight < 0)
{
  pwmRight= 0;
}
if(pwmRight > 255)
{
  pwmRight=255;
}
//Left
if(pwmLeft < 0)
{
  pwmLeft= 0;
}
if(pwmLeft > 255)
{
  pwmLeft=255;
}
```

Finalment, s'envien els polsos *PWM* corresponents al driver controlador de motors *L298N*.

```
analogWrite(ENA, pwmLeft);
analogWrite(ENB, pwmRight);
```

L'última part del codi és la funció auxiliar que s'utilitza per al càlcul de la mediana de les mesures dels sensors.

```
#define swap(a,b) a ^= b; b ^= a; a ^= b;
#define sort(a,b) if(a>b){ swap(a,b); }

int median(int a, int b, int c, int d, int e)
{
    sort(a,b);
    sort(d,e);
    sort(a,c);
    sort(b,c);
    sort(a,d);
    sort(c,d);
    sort(b,e);
    sort(b,c);
    // this last one is obviously unnecessary for the median
    //sort(d,e);

    return c;
}
```

12.4 Ajust dels paràmetres del controlador PID

Una vegada implementat el controlador PID al codi, compilat el programa i carregat a la placa Arduino, cal ajustar el valor de les 3 constants del controlador (*Kp*, *Ki* i *Kd*) per tal de que el funcionament del vehicle sigui correcte.

De manera general, es diu que el terme proporcional del controlador serveix per a controlar l'acció present. El terme integral serveix per a corregir les desviacions del passat. I en tercer lloc, el terme derivatiu col·labora en l'actuació davant de canvis en el futur.

En termes més tècnics, hi ha diversos factors que es tenen en compte a l'hora d'estudiar el comportament d'un controlador PID: estabilitat, velocitat, error estacionari i pertorbació de control. A continuació es descriuran amb més detall.

L'estabilitat d'un controlador PID se centra principalment en la capacitat que té el sistema per mantenir-se al voltant del valor definit pel *Setpoint* durant un període llarg de temps. El sistema és inestable quan oscil·la constantment prenent valors per damunt o per sota del *Setpoint*, cada vegada més allunyats. Les oscil·lacions estables no són desitjables però es poden solucionar. Les oscil·lacions inestables han de ser eliminades.

La velocitat d'un PID, és la rapidesa amb la que el sistema es col·loca en el punt desitjat de manera estable, partint d'una posició allunyada del *Setpoint*.

L'error estacionari és la diferència permanent entre el *Setpoint* i l'Input una vegada el sistema ja s'ha estabilitzat.

El terme pertorbació de control, fa referència a com queda el sistema després de que s'hi afegeixi una pertorbació a l'entrada. Si per exemple, el sensor d'entrada d'un sistema de control de temperatura d'un recipient, experimenta un cop de calor sobtat, l'estudi de l'afectació en l'*Output* degut a aquesta pertorbació correspon a la expressió "pertorbació de control".

Per ajustar manualment el valor de les constants del controlador PID, existeixen unes normes heurístiques d'ajust que mostren com varia el sistema en funció de la variació de cada una de les tres constants. A la taula 7 es pot observar com varia el sistema en funció de com canvien *Kp*, *Ki* i *Kd*.

	Kp augmenta	*Ki* augmenta	*Kd* augmenta
Estabilitat	Disminueix	Disminueix	Augmenta
Velocitat	Augmenta	Augmenta	Augmenta
Error estacionari	No eliminat	Eliminat	No eliminat
Pertorbació de control	Augmenta bruscament	Augmenta gradualment	Augmenta molt bruscament

Taula 7: Normes heurístiques d'ajust del controlador PID

Com es pot deduir de la taula, la cerca de les constants que facin òptim el funcionament del controlador i al seu torn del sistema complet, no és una tasca trivial. Si s'augmenta, per exemple, el valor de *Kd* per fer el sistema més estable, això pot provocar que davant una pertorbació el sistema reaccioni pitjor. Per tant, el que es guanya per una banda, es pot perdre per una altra. El mateix passa amb *Ki* i l'error estacionari. És obvi que si es vol eliminar, és obligatori incorporar l'acció integral al controlador. No obstant, aquest augment fa disminuir l'estabilitat. Per tant, en tots els valors de les constants, caldrà buscar un terme mig que permeti complir totes les especificacions de manera òptima.

Existeixen diverses tècniques d'ajust dels paràmetres del PID. Una de les més conegudes és el mètode clàssic d'ajust de Zieger i Nichols, autors que es basaren en la pràctica per a desenvolupar-lo.

Aquest mètode només és vàlid per a plantes estables en llaç obert (sense realimentació) i es duu a terme seguint els següents passos:

1. Utilitzant només un control proporcional (*Ki* i *Kd* fixades a 0), començant amb un valor de guany petit, incrementar aquest valor fins que el llaç comenci a oscil·lar. Cal que siguin oscil·lacions lineals i que aquestes siguin observades a la sortida del controlador.
2. Registrar el guany crític del controlador *Kp* = *Kc* i el període d'oscil·lació de la sortida del controlador, *Pc*.
3. Ajustar els paràmetres del controlador segons la taula 8.

	Kp	*Ki*	*Kd*
P	0,5·Kc	-	-
PI	0,45·Kc	(1,2·Kp) / Pc	-
PID	0,6·Kc	Kp / (0,5·Pc)	Kp·(Pc / 8)

Taula 8: Valor dels paràmetres del controlador segons el mètode de Zieger i Nichols

No obstant, a la pràctica, era difícil aplicar aquest mètode. En primer lloc era difícil discernir quan el sistema començava a oscil·lar perquè s'havia tornat inestable o perquè estava reaccionant a una pertorbació exterior. I en segon lloc, un cop començava a oscil·lar era difícil determinar amb exactitud el període d'oscil·lació.

El costat bo, és que gràcies al descobriment que van fer Zieger i Nichols, s'han desenvolupat més els mètodes manuals d'ajust, basant-se en el "prova i error". A arrel d'aquest fet, s'ha decidit finalment utilitzar una tècnica basada en el seu mètode però amb alguna variant.

1. Establir tots els guanys (*Kp*, *Ki* i *Kd*) a 0.
2. Augmentar el guany P fins que la resposta a una pertorbació sigui una oscil·lació estable.
3. Augmentar el guany D fins que les oscil·lacions desapareguin (esmorteïment crític).
4. Augmentar el guany I fins que el sistema s'apropi al *Setpoint* (eliminar l'error estacionari) amb el nombre d'oscil·lacions desitjades.

Els valors definitius als que s'ha establert el controlador PID són:

$$Kp = 3{,}7 \ ; \ Ki = 1{,}5 \ ; \ Kd = 0{,}3$$

Així doncs, s'ha aconseguit un sistema estable, centrat, amb una velocitat acceptable i un bon comportament davant l'entrada de pertorbacions. També s'ha aconseguit un sobrepuig petit. El sobrepuig és la quantitat que l'*Input* sobrepassa el *Setpoint* quan el sistema s'intenta estabilitzar. De vegades és pot millorar la velocitat del sistema a costa de tenir un sobrepuig més gran. En aquest cas s'ha optat per tenir un sobrepuig petit o gairebé inexistent, en front de millorar la velocitat.

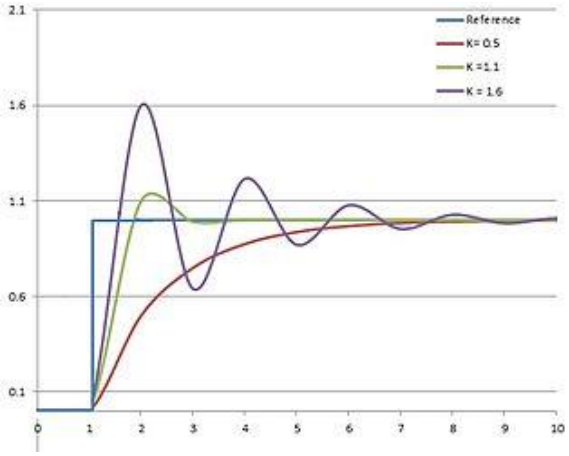


Figura 41: Sistemes amb molt sobrepuig (lila), sobrepuig moderat (verd) i sense sobrepuig (vermell)

13. Pressupost

A continuació es mostrarà el càlcul del cost de realització del projecte. Es detallarà el cost de cada component, de les eines utilitzades per desenvolupar l'aplicació, de l'amortització de l'ordinador utilitzat i de les hores dedicades.

Material			
Element	Quantitat	Preu Unitari (€)	Cost Total (€)
Arduino Uno R3	1	26,58	26,58
Sensors *HC-SR04*	2	2,68	5,36
Driver *L298N*	1	1,58	1,58
Xassís+rodes+motors	1	15,74	15,74
Cablejat i material electrònic	2	11,34	11,34
Suports per als sensors	2	1,15	2,3
Total			**62,9**

Taula 9: Costos de material del projecte

Despeses de personal			
Tipus de feina	Hores	Cost Horari (€/h)	Cos Total (€)
Estudi previ	80	20,00	1.600,00
Programació	120	40,00	4.800,00
Muntatge	10	10,00	100,00
Proves	75	40,00	3.000,00

Redacció memòria	80	40,00	3.200,00
Total			**12.700,00**

Taula 10: Costos de personal del projecte

Consum de recursos			
Tipus de recurs	Hores	Cost Horari (€/h)	Cost Total (€)
Consum elèctric del PC, plaques i bateria (100W)	270	0,14	37,80
Consum elèctric de l'espai de treball	270	0,14	37,80
Total			**75,60**

Taula 11: Costos de recursos del projecte

Cost total del projecte	
Part	Cost (€)
Material	62,9
Despeses de personal	12.700,00
Consum de recursos	75,60
Total	**12.838,5**

Taula 12: Cost total del projecte

14. Impacte ambiental

En aquest capítol s'estudia l'impacte ambiental associat al desenvolupament d'aquest projecte, en el que s'han fet servir diferents tipus de components electrònics i plàstics, i se'n avaluen les seves conseqüències.

Les opcions a considerar quan acaba la vida útil del producte són el reciclatge o el desfet en gestors de residus especialitzats en residus electrònics.

Si es considera el reciclatge de les parts d'un exemplar de sistema, aquest resulta inviable ja que la reutilització implicaria un cost més elevat que l'adquisició de nous components. Per tant, la millor opció consisteix en la reutilització de cada conjunt per a altres funcions.

Actualment, els aparells electrònics són cada vegada més abundants i això provoca al mateix temps un gran augment dels residus electrònics. Aquests residus contenen un alt grau d'elements tòxics que poden comportar grans riscos al medi ambient i a la salut pública, degut al tipus de materials de que estan fets, com ara el plom, l'arsènic, el mercuri, el coure, etc. A més, tot i que es facin diferents fases de reciclatge, és necessari acabar desfent els components de forma correcta. Això es realitza en centres de gestió de residus d'aparells elèctrics i electrònics (RAEE).

Per això, s'ha de prendre consciència de la importància de reciclar adequadament cada un dels seus components quan queden en desús.

Afortunadament, la legislació ambiental és cada vegada més exigent i les persones estan més conscienciades del grau d'importància del reciclatge, i en concret, també del reciclatge de components electrònics. Per això existeixen diferents punts de recollida d'aquests materials que s'encarregaran de fer un tractament dels residus. L'objectiu principal d'aquest propòsit és reduir les quantitats de residus i la perillositat dels seus components i també regular la gestió per millorar la protecció ambiental, fomentant la seva reutilització i valoració.

15. Conclusions

En aquest treball de fi de grau s'ha proposat dissenyar i construir una maqueta d'un vehicle mòbil autònom controlat per una placa Arduino que pogués circular pel centre d'una trajectòria delimitada per parets, mantenint tota l'estona una distància igual a dreta i esquerra. Ha estat una experiència molt enriquidora i valuosa, degut a que es desconeixia com procedir a la realització d'uns sistema completament autònom que permetés el control dels motors en funció de la informació proporcionada per uns sensors.

S'han aconseguit tots els objectius generals que es proposaven en aquest projecte. El muntatge del vehicle ha estat la part més fàcil, ja que un cop trobats tots els components, es tenia molt clar com havien d'anar integrats. La programació del codi que fa funcionar el vehicle i integra totes les parts ha estat la part més complicada.

Sens dubte, la part més difícil de tot el projecte ha estat la implementació del controlador PID, especialment l'ajust dels seus paràmetres. Inicialment es van intentar ajustar-lo a mà, com finalment s'ha fet. No obstant, en un inici els sensors no disposaven de cap suport i estaven col·locats de manera desigual. Degut al petit angle de detecció de què disposen, al no estar situats igual, els resultats que proporcionaven no eren del tot comparables, i això portava problemes en l'*Input* del PID. Per a solucionar-ho es van comprar uns suports que servien per fixar els sensors al xassís del vehicle.

Una vegada aconseguida la fixació dels sensors, es va veure que calia tractar de manera adequada les seves mesures, ja que podien proporcionar resultats no fiables. En aquest sentit es van tractar les dades com a nombres enters en comptes de decimals, i es va instal·lar un filtre que mitigués una mica el soroll en les mesures.

Quan ja es tenien mesures bones i fiables, va començar l'àrdua feina d'ajustar els paràmetres. Es va intentar incloure en el programa una llibreria anomenada *AutoTune* que ajusta els paràmetres de manera automàtica. Encara que els resultats finals que proporciona no són els definitius, la llibreria ofereix uns valors de base des dels quals es pot començar l'ajust. A més, prenent referència d'altres projectes fets amb Arduino que inclouen PIDs, es va desenvolupar un programa en un altre entorn que permetés veure en un gràfic i modificar els paràmetres del controlador en temps real. Aconseguir fer funcionar aquest programa va ser una feina complicada d'assolir, i malauradament al final no es va fer servir per ajustar, perquè no aportava millores significatives. Tampoc es va fer servir la llibreria *AutoTune* perquè no acabava de funcionar en el vehicle, pel tipus de control PID que presenta, ja que l'*Input* i l'*Output* estan lligats a entorns diferents, un als sensors i l'altre als motors.

Finalment, després de veure molts vídeos i llegir molts articles en webs de projectes amb PID en Arduino i de comprendre millor el funcionament intern del controlador, es va aconseguir un ajust acceptable de les tres constants del PID.

Com a futures sofisticacions del projecte, es podrien afegir un seguit de sensors més a la part davantera del vehicle, que permetessin tenir una visió més completa dels voltants. Actualment, l'angle de detecció de que disposen els sensors és petit, i això fa que el vehicle només pugui circular per trajectòries suaus, amb curvatures no gaire pronunciades.

També es podria optimitzar més l'ajust dels paràmetres del controlador PID. Es podria també intentar modelitzar matemàticament el funcionament del vehicle, i aplicar els coneixements adquirits a les assignatures de Dinàmica de Sistemes i Control Automàtic per aconseguir un ajust més òptim dels paràmetres mitjançant la simulació.

Cal destacar també que l'objectiu principal d'aprofundir els coneixements en la branca d'electrònica s'ha assolit amb satisfacció. La lectura de nombroses fulles de característiques dels components i dels esquemàtics ha permès fer possible aquest fet. A més, també s'ha ampliat el coneixement sobre els microcontroladors, el seu funcionament, les seves aplicacions i les seves limitacions. De retruc i sense ser una de les intencions principals, també s'ha aprofundit en els coneixements dels sistemes de control, gràcies a la implementació del controlador PID.

També s'ha aconseguit aprofundir coneixements en programació en un llenguatge que no s'aprèn en el Grau d'Enginyeria en Tecnologies Industrials.

Bibliografia

-Arduino. *Arduino Uno Rev3 Schematic*, 2010.

-Atmel Corporation. *ATmega328/P Datasheet*, 2016.

-Brett Bauregard. *Archive for the 'Front End' Category* (4 de Juliol de 2009). Recollit de Brett Bauregard: http://brettbeauregard.com/blog/category/pid/front-end/

-Brett Bauregard. *Arduino PID Library* (sense data). Recollit d'Arduino: https://playground.arduino.cc/Code/PIDLibrary

-Brett Bauregard. *Improving the beginner's PID* (sense data). Recollit de Brett Bauregard: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

-Brett Bauregard. *Processing Front-End for the PID Library* (30 de Maig de 2009). Recollit d'Arduino: https://forum.arduino.cc/index.php?topic=6998.0

-Brian W. Evans. *Arduino Notebook: A Begginer's Referece Written*, 2007.

-David Pilling. *HC-SR04* (30 de Desembre de 2015). Recollit de: https://www.davidpilling.com/wiki/index.php/HCSR04

-Elecfreaks. *Ultrasonic Ranging Module HC-SR04 Datasheet*, 2012.

-Fernando Morilla García. *El controlador PID*. UNED, 2007.

-*HC-SR04: tests on accuracy, precision and resolution of ultrasònic measurement* (27 de Maig de 2014). Recollit d'Arduino: https://forum.arduino.cc/index.php?topic=243076.0

-*Kalman filter vs Complementary filter* (sense data). Recollit de http://robottini.altervista.org/kalman-filter-vs-complementary-filter

-*La referencia analógica AREF* (sense data). Recollit de Prometec: https://www.prometec.net/aref/

-Luis Llamas*. Opciones para alimentar Arduino con bateries* (5 de març de 2016). Recollit de Luis Llamas: https://www.luisllamas.es/alimentar-arduino-baterias/

-ON Semiconductor. *NCP1117, NCV1117 Datasheet*, 2011.

-Sharp. *GP2Y0A21YK/GP2Y0D21YK General Purpose Type Distance Measuring Sensors Datasheet*, 2010.

-ST Microelectronics. *L298 Dual Full-Bridge Driver Datasheet*, 2000.

-Tim Eckel. *NewPing Library for Arduino* (sense data). Recollit de Arduino: http://playground.arduino.cc/Code/NewPing

-Vt en linea. *Como usar el driver controlador de motores L298N en Arduino* (4 de Setembre de 2017). Recollit de Como usar el driver controlador de motores L298N en Arduino: https://www.youtube.com/watch?v=0bxqxp9EkVI

Agraïments

Vull donar les gràcies al professor Emili Lupon, del departament d'Enginyeria Electrònica, per la dedicació, guiar-me i aconsellar-me al llarg del projecte.

També vull agrair a la meva família, sobretot als meus pares i als meus germans, i als amics per tot el seu suport, paciència i comprensió durant els meus estudis i especialment en el treball de fi de grau.

Annex

A: Programa final del projecte

```
#include <Array.h>
#include <PID_v1.h>
#include <NewPing.h>



//MotorA
int IN1 = 8; //gris
int IN2 = 7; //marron
int ENA = 9; //rodes esquerra // negre
//MotorB
int IN3 = 13; //blau
int IN4 = 12; //vermell
int ENB = 11; //rodes dreta // verd
//Sensor Ultraso Dreta
int TrigR = 5; //Cable marron
int EchoR = 3; //Cable vermell
//Sensor Ultaso Esquerra
int TrigL = 2; //Cable Lila
int EchoL= 4; //Cable taronja


double Setpoint, Input, Output;
double Kp=3.7, Ki=1.5 , Kd=0.3;
int velocitat_mitja = 125;
int diferencia_distancia;
unsigned long fdistR=0, fdistL=0;
int diferencia_velocitat;
```

```
const byte size = 5;

//creem llista dreta
int raw_dreta_array[size]={0,0,0,0,0};
Array<int> llista_dreta = Array<int>(raw_dreta_array,size);

//creem llista esquerra
int raw_esquerra_array[size]={0,0,0,0,0};
Array<int> llista_esquerra = Array<int>(raw_esquerra_array,size);

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

NewPing sonarR(TrigR,EchoR,300);
NewPing sonarL(TrigL,EchoL,300);

unsigned long serialTime; //this will help us know when to talk with processing

void setup() {
  Serial.begin(9600);
  pinMode(TrigR, OUTPUT);
  pinMode(EchoR, INPUT);
  pinMode(TrigL, OUTPUT);
  pinMode(EchoL, INPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
```

```
  pinMode(ENB, OUTPUT);

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

  Setpoint = 0;
  //iniciem el PID
  myPID.SetMode(AUTOMATIC);
  double limit_inferior=-velocitat_mitja;
  double limit_superior=velocitat_mitja;
  myPID.SetOutputLimits(limit_inferior, limit_superior);

  delay(4000); //deixem 4s perquè s'inicii i es coloqui tot

}

void loop()
  {
  int distanciaR = ((sonarR.ping()*10)/58.4)+0.5;
  int distanciaL = ((sonarL.ping()*10)/58.4)+0.5;

  //afegim a la llista dreta
  llista_dreta[0]=llista_dreta[1];
  llista_dreta[1]=llista_dreta[2];
  llista_dreta[2]=llista_dreta[3];
  llista_dreta[3]=llista_dreta[4];
  llista dreta[4]=distanciaR;
```

```
//afegim a la llista esquerra
llista_esquerra[0]=llista_esquerra[1];
llista_esquerra[1]=llista_esquerra[2];
llista_esquerra[2]=llista_esquerra[3];
llista_esquerra[3]=llista_esquerra[4];
llista_esquerra[4]=distanciaL;

fdistR=median(llista_dreta[0],llista_dreta[1],llista_dreta[2],llista_dreta[3],llista_dreta[4]);
fdistL=median(llista_esquerra[0],llista_esquerra[1],llista_esquerra[2],llista_esquerra[3],llista_esq

/*Serial.print("\t");
Serial.print(fdistR);
Serial.print("\t \t");
Serial.print("\t"); //per al serial.plotter
Serial.print(fdistL);
Serial.print("\n");*/

diferencia_distancia= fdistR-fdistL;
Input = diferencia_distancia;

myPID.Compute();

diferencia_velocitat = Output;


int pwmRight = velocitat_mitja + diferencia_velocitat;
int pwmLeft = velocitat_mitja - diferencia_velocitat;
```

```
if(pwmRight < 0)
{
  pwmRight= 0;
}
if(pwmRight > 255)
{
  pwmRight=255;
}
//Left
if(pwmLeft < 0)
{
  pwmLeft= 0;
}
if(pwmLeft > 255)
{
  pwmLeft=255;
}

Serial.print("\t");
Serial.print(pwmRight);
Serial.print("\t \t");
Serial.print("\t"); //per al serial.plotter
Serial.print(pwmLeft);
Serial.print("\n");

analogWrite(ENA, pwmLeft);
analogWrite(ENB, pwmRight);


#define swap(a,b) a ^= b; b ^= a; a ^= b;
#define sort(a,b) if(a>b){ swap(a,b); }

int median(int a, int b, int c, int d, int e)
{
    sort(a,b);
    sort(d,e);
    sort(a,c);
    sort(b,c);
    sort(a,d);
    sort(c,d);
    sort(b,e);
    sort(b,c);
    // this last one is obviously unnecessary for the median
    //sort(d,e);

    return c;
}
```